

C:\NtPortbl\PALib1\CB9\1210448 1.DOC

Method and Apparatus for Creating a Well-Formed Database System Using a Computer

Inventors: Craig David Weissman, Greg Vincent Walsh and Eliot Leonard Wegbreit

Copyright Notice

5 A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by any one of the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

The Field of the Invention

10 This invention relates to the field of databases. In particular, the invention relates to creating databases, and loading and accessing data in the databases.

Background of the Invention

15 Many different types of databases have been developed. On line transaction processing (OLTP) databases are examples of typical databases used today. OLTP databases are concerned with the transaction oriented processing of data. On line transaction processing is the process by which data is entered and retrieved from these databases. In these transaction-oriented databases, every transaction is guaranteed. Thus, at a very low level, the OLTP databases are very good at determining whether any specific transaction has occurred.

20 Another type of database is a data warehouse or datamart. A datamart transforms the raw data from the OLTP databases. The transformation supports queries at a much higher level than the OLTP atomic transaction queries. A data warehouse or a datamart typically provides not

only the structure for storing the data extracted from the OLTP databases, but also query analysis and publication tools.

The advantage of datamarts is that users can quickly access data that is important to their business decision making. To meet this goal, datamarts should have the following

5 characteristics. First, datamarts should be consistent in that they give the same results for the same search. The datamart should also be consistent in the use of terms to describe fields in the datamart. For example, "sales" has a specific definition, that when fetched from a database, provides a consistent answer. Datamarts should also be able to separate and combine every possible measure in the business. Many of these issues are discussed in the following book,
10 Ralph Kimball, *The Data Warehouse Toolkit*, John Wiley and Sons, Inc., New York, NY (1996).

Multi-dimensional datamarts are one kind of datamart. Multi-dimensional datamarts rely on a dimension modeling technique to define the schema for the datamart. Dimension modeling involves visualizing the data in the datamart as a multi-dimension data space (e.g., image the data
15 as a cube). Each dimension of that space corresponds to a different way of looking at the data. Each point in the space, defined by the dimensions, contains measurements for a particular combination of dimensions. For example, a three dimensional cube might have product, customer, and territory dimensions. Any point in that cube, defined by those three dimensions, will represent data that relates those three dimensions.

20 The data in the datamart is organized according to a schema. In a dimensional datamart, the data is typically organized as a star schema. At the center of a standard star schema is a fact table

that contains measure data. Radiating outward from the fact table, like the points of a star, are multiple dimension tables. Dimension tables contain attribute data, such as the names of customers and territories. The fact table is connected, or joined, to each of the dimension tables, but the dimension tables are connected only to the fact table. This schema differs from that of many conventional relational databases where many tables are joined. The advantage of such a schema is that it supports a top down business approach to the definition of the schema.

Present datamarts have a number of drawbacks that are now discussed. First, datamarts are typically difficult to build and maintain. This is because of the requirements that they be consistent and flexible. A related drawback of present day datamarts is that they do not allow the consultants of the datamart to make changes to the schema simply and easily. Because datamarts support very high level queries about the business processes in the business, they require a great deal of consistency in the use of data from the OLTP systems. Additionally, the datamarts need to be very flexible to address changes in the types of high level queries supported. Changing typical datamarts require the changing of hundreds, or potentially thousands, of lines of SQL code. For example, if a fact column is added to a fact table, the change propagates throughout the datamart. These changes are typically implemented by hand, a very time consuming and error prone process. As a result of the hand coding involved, it is quite possible to construct the database in an arbitrary fashion that does not conform to good rules for constructing datamarts. Thus, well-formed datamarts may not result.

Thus an improved data warehousing technology is desired.

A Summary of the Invention

One embodiment of the invention includes a method of defining a well-formed database system by defining the organization of the data in the database, and by defining the operations for that data. The definition can then be used to automatically create and populate the well-formed database system. The well-formed database system conforms to rules of correctness and produces results that conform to the rules. The organization is defined by a data organization definition that specifies tables, their columns, and the relationships between tables. The operations define procedures that operate on the tables and the table columns. Importantly, the operations are defined along with the tables, columns, and relationships, so that the resulting system is well-formed. Without this invention, database systems can be constructed in an arbitrary and inconsistent fashion which can result in an incorrectly constructed database system.

In some embodiments, when the database system is created, it automatically includes the following capabilities: foreign key tracking, automatic indexing, time and date information inclusion. By including some or all of such capabilities in the database system, the system will operate to comply with the rules of correctness.

The following are aspects of various embodiments of the invention. The constructed well-formed database system can automatically guarantee the following. (1) Two columns related by a relational join will be from the same domain. (2) If table A has a many-to-one relationship to table B, then table A has a foreign key that corresponds to table B. (3) A many-to-many relationship, between two tables A and B, is always expressed by an associative table that is created in a uniform way. For each unique many-to-many relationship, a unique value is created

in the associative table and reused whenever that many-to-many relationship occurs.

Denormalization is always done correctly. (4) Pulling information from one table to be put into another table, for access efficiency, is done correctly.

In some embodiments of the invention, the data organization definition includes a schema
5 description for a datamart. The datamart automatically includes the inclusion of transaction type information and the mapping of source system keys. In these embodiments, the operation definitions define one or more of the following sets of operations: datamart population operations, aggregate creation and maintenance operations, query and result interface operations.

Although many details have been included in the description and the figures, the invention is
10 defined by the scope of the claims. Only limitations found in those claims apply to the invention.

A Brief Description of the Drawings

The figures illustrate the invention by way of example, and not limitation. Like references indicate similar elements.

Figure 1 illustrates a datamart system representing one embodiment of the invention.

5 Figure 2 illustrates an embodiment of a method of defining the datamart, loading the datamart, and then querying the data.

Figure 3 illustrates a schema used in the system of Figure 1 to define schemas for the datamart.

Figure 4 illustrates a schema used in the data extraction and loading process.

10 Figure 5 illustrates a runtime schema including aggregates.

Figure 6 illustrates a query mechanism and user interface schema.

Figure 7 through Figure 29 describe a user interface that can be used to define a schema, build a datamart, load the datamart, and query the datamart.

15 Figure 30 through Figure 36 describe a user interface that can be used by a consultant to set up the query interface for a user and to provide the reporting interface.

The Description

Table of Contents

	COPYRIGHT NOTICE	1
	THE FIELD OF THE INVENTION	1
5	BACKGROUND OF THE INVENTION	1
	A SUMMARY OF THE INVENTION.....	4
	A BRIEF DESCRIPTION OF THE DRAWINGS.....	6
	THE DESCRIPTION	7
	TABLE OF CONTENTS.....	7
10	INTRODUCTION TO THE DESCRIPTION.....	8
	DEFINITIONS	10
	DATAMART SYSTEM.....	11
	<i>System Element List</i>	12
	<i>System Element Descriptions</i>	13
15	Metadata Overview	13
	System Overview	15
	External Elements.....	17
	EXAMPLE METHOD OF DEFINING AND USING THE DATAMART	19
	TOP LEVEL METADATA SCHEMA	23
20	<i>Top Level Metadata List</i>	24
	<i>Top Level Metadata Descriptions</i>	25
	Fact Related Tables	25
	Dimension Related Tables.....	27
	Semantic Instance Table.....	30
25	Aggregate Related Tables.....	30
	Data Store Related Tables	32
	Cleansing Related Tables.	33
	Additional Tables	33
	<i>Top Level Metadata Use</i>	34
30	EXTRACTION METADATA	36
	<i>Extraction Metadata List</i>	36
	<i>Extraction Metadata Descriptions</i>	37
	Job Related Tables.....	37
	Connector Related Tables.....	39
35	Extraction Group Related Tables	41
	SQL Statement Related Tables.....	42
	Error Handling.....	43
	Data Semantic Related Tables.....	43
	Data Store Related Tables	45
40	<i>Extraction Metadata Use</i>	47
	<i>Further Discussion of Templates</i>	49

	<i>Examples</i>	50
	Additional Templates	51
	RUNTIME METADATA	53
	<i>Runtime Metadata List</i>	55
5	<i>Runtime Metadata Descriptions</i>	55
	<i>Time Navigation</i>	60
	QUERY MECHANISM METADATA	62
	<i>Query Mechanism Schema List</i>	63
	<i>Query Mechanism Schema Metadata Descriptions</i>	63
10	Ticksheets Metadata	63
	Measurement Metadata.....	66
	Filtering Metadata	67
	Display Options Metadata	69
	USER INTERFACE EXAMPLE OF DEFINING METADATA	70
15	<i>General Schema Definitions User Interface</i>	70
	<i>Extraction Interface Elements</i>	73
	<i>Additional Interface</i>	75
	<i>End User Interface Definition and Example</i>	76
	ALTERNATIVE EMBODIMENTS	83
20	THE CLAIMS	86
	THE ABSTRACT	98
	APPENDIX A	99

Introduction to the Description

The following describes a system according to various embodiments of the invention.

Generally, the system allows a consultant to define a well-formed datamart. The system includes tables and columns that conform to the definition of the datamart. The system also includes additional columns for foreign key tracking, source system key mapping, time and date tracking.

The system has automatic indexing. The system enforces typing information about the data stored in the datamart. These additional features cause the datamart to operate in a consistent manner. One benefit of such consistent operation is that results are consistent in meaning from query to query.

Focusing on the datamart creation, the system allows a consultant to build a datamart from a schema definition and a definition of the sources of the data. From the schema definition, the system automatically builds the tables needed in the datamart. Also, from the schema definition, and the sources definition, the system can automatically extract the data from those sources.

5 Depending on the semantic meaning of the data, as defined by the schema definition, the system automatically converts the data from the sources into forms that are readily usable in the datamart. Once the datamart has been created, and the data has been loaded, users can then perform queries on the data.

As part of the datamart creation, the system allows the consultant to define aggregates for the datamart. The aggregates correspond to pre-computed query results for different types of queries. For example, an aggregate can be created for a query that asks for all sales, by region, by quarter. The corresponding aggregate table would include a set of rows that have the results for this query (e.g., each row includes the quarterly sales for each region). The aggregates are specified using the schema definition. This makes defining and changing aggregates relatively simple.

To allow a user to query the datamart, the system includes an interface for defining what fields can be used by the user to query the datamart. Additionally, by allowing the consultant to define measure and related information, the system allows the consultant to specify how the results are to appear to the users.

20 The following description first presents a system level view of primarily one embodiment. Then, an example use of the system is presented. Next, the metadata used in the system is

described. This metadata description is broken into four parts: a top level description of the metadata used in defining schemas, a description of the metadata used during the extraction, a description of the metadata used while the datamart is running, and a description of the query interface metadata. Next, an example set of user interface screen shots illustrates how consultants can quickly and efficiently define schemas, aggregates, and query interfaces, and how users can query the datamart. Next, additional alternative embodiments are described.

Definitions

Datamart or Data Warehouse – is a database.

Schema – is a description of the organization of data in a database. Often, the schema is defined using a data definition language provided by a database management system. More abstractly, the schema can be the logical definition of a data model for use in a database.

Metadata – is data that defines other data. This is not the actual data in the datamart, but is the data that defines the data in the datamart.

Constellation – a grouping of dimension definitions, fact definitions, like-structured facts (all facts in a constellation have the same dimensional foreign keys), or stars, and other metadata definitions. Often the grouping relates to a business process (e.g., sales).

Fact Table – the central table of a star schema. It stores the numeric measurements of the business that is supplying the information to the datamart.

Measurement – is a piece of data in a fact table, or an arithmetic combination of data.

Dimension – the tables that link to the fact table in a star schema. The tables store the descriptions of the dimensions of the business. Examples of dimensions are product and territory.

Attributes – are the fields of a dimension table (e.g., product name, country name).

5 User – any end user who would normally wish to query a datamart, but would not usually be concerned with the implementation or maintenance of the datamart.

Consultant – is a person responsible for the creation and maintenance of a datamart.

Source System – is any computer system that holds the raw data used by the system.

Examples of such source systems are OLTP database systems.

10 Data Store – any data storage (physical or logical) from which data is received or to which data is stored. Examples of a data store are files, a database, etc.

Computer – is any computing device (e.g., PC compatible computer, Unix workstation, etc.).

Generally, a computer includes a processor and a memory. A computer can include a network of computers.

15 Program – a sequence of instructions that can be executed by a computer. A program can include other programs. A program can include only one instruction.

Datamart System

Figure 1 illustrates a datamart system representing one embodiment of the invention. The system supports the creation of a well-formed datamart. This system allows consultants to use
20 metadata to define schemas for a datamart. From the definition of the schema, the system can automatically generate the tables in the datamart. Further, the system can automatically extract

the data from the source systems, perform conversions on that data and populate the datamart.

The system supports the automatic creation and processing of aggregates from aggregate definitions. The system also supports the creation of the query mechanisms from query definitions.

5 The following description first lists all the elements of Figure 1, then describes each of those elements, and then discusses how those elements operate together.

System Element List

Figure 1 includes the following elements: source systems 110, a system 100, a web server 186, a consultant computer 190, and a user computer 180. The system 100 includes the metadata 160, an enterprise manager 102, an extraction program 120, staging tables 130, a semantic
10 template conversion program 140, a datamart 150, an aggregate builder 170, and a query and reporting program 104. The metadata 160 includes the following data: schema definitions 161, connectors 162 (connectors are also referred to as extractors), semantic definitions 163, source system information 164, aggregate information 167, measurement information 168, and
15 query/reporting information 169. The user computer 180 is shown running a browser 182. The browser 182 includes a query/results interface 184. The consultant computer 190 shows the enterprise manager interface 192 which shows the metadata organization of the system 100.

System Element Descriptions

The following describes the metadata 160, then the other elements of the system 100, and finally, the elements that are external to the system 100. These elements are all described in greater detail below.

Metadata Overview

The metadata 160 includes many different types of data and information. This information can be broken down into information related to (1) the definition of the schema for the datamart 150, (2) the data needed during the extraction from the source systems 110 and loading of the datamart 150, and (3) the information used in the querying of the datamart 150 and supplying the result sets. The relationships between the elements of the metadata 160 are described in greater detail below. However, the following provides brief descriptions of these elements.

The schema definitions 161 hold the definition of the schema for the datamart 150.

Typically, a consultant, using the consultant computer 190, can interface with the enterprise manager 102 to define the schema definition 161 for the datamart 150. In particular, the consultant can use the enterprise manager interface 192 to define a star schema for the datamart 150. This star schema is organized around the business processes of the business for which the datamart is being created. What is important is that the consultant can easily define a schema for the datamart 150 and that definition is kept in the schema definitions 161. From the schema definitions 161, not only can the tables in the datamart 150 be generated, but also the automatic extraction and conversion of the data from the source systems 110 can be performed, aggregates are set up, and a query mechanism is generated.

The connectors 162, the semantic definitions 163, and the source system information 164, are all related to the extraction of the data from the source systems 110. The connectors 162 define the access routines for extracting the source systems data 110. The semantic definitions 163 define how that extracted data should be converted when it is loaded into the datamart 150. The semantic definitions 163 provide important advantages to the system 100. In particular, the semantic definitions 163 allow for a simplified definition of the datamart 150, consistent meaning of the data in the datamart 150, and allow for complex changes to the schema to be easily propagated to the datamart 150. The source system information 164 defines how to extract the data from the systems 110.

The aggregate information 167 defines how data in the datamart 150 is treated once it is extracted. The aggregate information 167 allows for the creation of aggregates. Aggregates are aggregations of various fields of data in the datamart 150. Aggregates support more complex and powerful queries to be executed on the datamart 150. The aggregates also improve the performance of the system during the querying process and allow for time navigation of the data in the datamart 150. Time navigation is the process of creating backlog result sets by hopping through date aggregates from the beginning of time in the datamart 150 to the present.

The measurement information 168 and the query/ reporting information 169 support the querying of the datamart 150. A measure is a piece of numeric data in the datamart 150 that is useful to a user. That is, individual fact columns from source systems can be very implementation specific. These columns may not correspond to what users would prefer to see. For example, a user may want to see a net price added with a total cost. However, the fact table

may only include the net price or the total cost. The measure information 168 allows the consultant to define the abstract notion of the calculation associated with the net price added to the total cost.

In some embodiments of the invention, the metadata 160 also includes security information.

5 The security information defines the level of access for various users to the various tables and fields in the datamart 150. This security information automatically restricts access to that data.

System Overview

The system 100 can be implemented on a network of computers running Windows NT and UNIX. The datamart 150 can be implemented on top of an Oracle (SQL Server, or ODBC)
10 database. However, this physical structure of the system 100 can be implemented in any number of ways, and the invention does not require this specific hardware configuration.

The enterprise manager 102 is a program that is responsible for supporting the definition of the schema, and the creation of the tables in the datamart 150 from the schema definitions 161. The enterprise manager 102 also controls the extraction program 120. (In some embodiments,
15 the extraction program 120 and the semantic template conversion program 140 are included in the enterprise manager 102). During the execution of the extraction program 120, the extraction program 120, the staging tables 130, the semantic template conversion 140, and the datamart 150 are all used. The extraction program 120 uses the connectors 162 and the source system information 164 to extract the information from the source systems 110. The extracted data is
20 loaded into the staging tables 130.

The staging tables 130 are temporary tables used to hold the source system data before performing any semantic conversions on that data. The staging tables 130 also allow for the conversion of the source system data prior to moving the data into the datamart 150.

Once the staging tables 130 have been loaded, the semantic definitions 163 can be accessed
5 from the enterprise manager 102 to convert the information in the staging tables 130 to predefined data semantics. These predefined data semantics allow for powerful queries, consistency in the definition of the meaning of the data in the datamart 150, and allow for changes to be made to the schema. Generally, the semantic template conversion 140 takes data stored in the staging tables 130, performs a conversion of that data according to a corresponding
10 semantic definition (defined in the schema definitions 161), and populates the datamart 150 with the converted data.

Importantly, the predefined data semantics substantially simplify the creation and population of the datamart 150. In previous systems, the consultant would have to implement all of the data manipulation and population programs by hand. By selecting a particular semantic definition for
15 a particular fact, or dimension, in the schema, the consultant has automatically defined the access and manipulation for populating programs for that table. Allowing the consultant to select a predefined data semantic not only reduces the tedious coding previously required of the consultant, but also allows for the automatic insertion of foreign keys, transaction types, date, and other information into the schema, and therefore the datamart 150. This additional
20 information causes the datamart 150 to be well-formed.

The aggregate builder 170, as mentioned above, aggregates data in the datamart 150 according to the aggregate information 167 and the schema definitions 161. The results of the aggregate builder 170 allow for more powerful and faster queries to be performed on the datamart 150.

5 The query/reporting program 104 supports the querying of the datamart 150 and presents results of those queries. The query and reporting process 104 uses the measurement information 168 and the query and reporting information 169, in addition to the schema definitions 161, to query the datamart 150 and provide that information to the web server 186. The query/reporting information 169 includes filters and form definitions. The filters allow the user to filter different fields out of the datamart 150. The forms allow the users to indicate which fields a user is particularly interested in.

15 The metadata 160, although including many different types of definitional data, importantly includes the schema definition 161 and the semantic definitions 163. The enterprise manager 102 can use the schema definitions 161 to build the tables in the datamart 150. Through the combination of these two pieces of metadata 160, the enterprise manager 102 can take data from a source system 110, perform semantic conversions on that data and populate the datamart 150. Thus, in some embodiments of the invention, the system includes only the schema definitions 161 and the semantic definitions 163.

External Elements

20 The source systems 110, as defined above, represent large databases from which data for the datamart 150 is pulled. Examples of such systems include large on line transaction processing

(OLTP) systems. Typically these source systems 110 are relational databases having multiple tables and relations between those tables. The source systems 110 do not generally support powerful queries that provide high level information about the business in which the source systems 110 are used. Thus, the system 100 is used to extract the data from the source systems 110 and to provide an improved schema for querying that data. In some embodiments, the source systems 110 include non-relational databases such as object databases. In other embodiments, the source systems 110 can include flat file systems or combined relational and object databases. What is important is the source systems 110 can provide data to the system 100 through the connectors 162 and the source system information 164.

The consultant computer 190 represents any computing device that allows a consultant to access the system 100. (Access to the system 100 can be through a network.) What is important is that the consultant computer 190 allows the consultant to interface with the enterprise manager 102. Note, that the enterprise manager 102 can run on the consultant computer 190.

The user can access the web server 186 through the user computer 180. In this example, the user computer 180 can access the web server 186 through an HTTP connection. The user computer 180 sends a file request to the web server 186. This file request represents a request for a query of the datamart 150. The web server 186 runs a Java program upon receiving the request. The query and reporting program 104 converts the information from the Java program into a query that the datamart 150 will understand. In one embodiment, the query/reporting program 104 converts the query into a set of SQL statements. The SQL statements are run

against the datamart 150. The results of the statements are processed and provided back to the user computer 180.

Example Method of Defining and Using the Datamart

Figure 2 illustrates an embodiment of a method of defining the datamart 150, loading the datamart 150, and then accessing the data in the datamart 150. This example can be broken into four subparts: a build datamart process 202, an extraction and loading process 204, a build aggregates process 205, and a query and reporting process 206. This example can be implemented using the system 100.

At block 210, a consultant uses the enterprise manager 102 to define the schema. The schema is defined using the metadata 160. This process is illustrated in greater detail in Figure 7 through Figure 35. Generally, defining the schema involves determining the business processes of the organization for which the system 100 is being implemented. The consultant then defines the star schema for those business processes. The star schema has a fact table and a number of dimensions. The consultant also defines from where the data in the schema is to be derived. That is, the consultant defines from which fields and tables the information is to be extracted from the source systems 110. The consultant also defines how that data is to be put into the datamart 150. That is, the consultant associates each piece of data with a semantic meaning. This semantic meaning defines how the data from the source system is to be manipulated and how it is to populate the datamart 150. At this point, the consultant can also define the aggregates that can be used in the datamart 150.

Once the datamart 150 has been defined, it can then be automatically built. At block 220, the enterprise manager 102 generates table creation SQL statements according to the definition of the metadata. In one embodiment of the invention, block 220 is accomplished by performing queries on the schema definitions 161 to generate the fact table creation statements, the fact staging table creation statements, the dimension table creation statements, the dimension staging table creation statements, and the dimension mapping table creation statements. These tables are described in greater detail below. From the results of these queries, SQL CREATE TABLE statements are created. Importantly, the schema definitions 161 provide the information the enterprise manager 102 needs to build the datamart 150.

Note that this process can also be used to modify the schema of an existing datamart 150. Therefore, at block 220, the SQL tables being created will cause the existing datamart 150 to be modified without losing the data in the datamart 150.

At block 230, the enterprise manager 102 issues the table generation statements to the database upon which the datamart 150 is being created. That database creates the tables, which correspond to the datamart 150. After block 230, the build the datamart process 202 is complete.

Now the extraction process 204 can be performed. The extraction process 204 is run on a periodic basis to load data from the source systems 110 into the datamart 150. This process can be run multiple times for the datamart 150.

At block 260, the connectors 162 are used by the enterprise manager 102, and in particular, they are used by the extraction program 120 to extract the data from the source systems 110. The connectors 162 can include SQL statement templates (not to be confused with semantic

templates, as described below) for extracting data from the source systems 110. The extraction program 120 uses these templates, in addition to the source system information 164, to generate SQL statements. These SQL statements are issued to the source system 110 and the results are loaded into the staging tables 130. (The staging tables 130 had been created as a result of block 230.) Once the staging tables have been loaded, the data can then be moved into the datamart 150.

At block 270, the staging table data is moved into the datamart 150 using the semantic definitions 163. The semantic definitions 163 are templates for converting the staging tables 130 data according to predefined data semantics. These predefined data semantics, as described below, provide semantic meaning to the data being loaded from the staging tables 130. Note that the data from the staging tables 130, as processed by the semantic template conversion 140, is placed in the tables in the datamart 150.

Thus, the schema definition and the semantic definitions 163 are used to generate and populate the datamart 150 such that the datamart 150 is well-formed. Examples of the well-formedness of the datamart 150 are as follows. (1) Two columns related by a relational join will be from the same domain. (2) If table A has a many-to-one relationship to table B, then table A has a foreign key that corresponds to table B. (3) A many-to-many relationship, between two tables A and B, is always expressed by an associative table that is created in a uniform way. For each unique many-to-many relationship, a unique value is created in the associative table and reused whenever that many-to-many relationship occurs. Denormalization is always done correctly. (4) Pulling information from one table to be put into another table, for access

efficiency, is done correctly. Previous systems cannot guarantee such a well-formed database system because hand coding of the creation and population operations is required. This hand coding can easily introduce errors into datamart creation and population processes.

Once the extraction process 204 has completed, the aggregates can be built in the build aggregates process 205. The aggregates are tables of pre-calculated combinations of dimensions and facts. Importantly, they greatly increase the speed of queries. Generally, the aggregate definitions, stored in the aggregate information 167, are accessed and built using the aggregate definitions (which interface with the schema definitions). At block 275, the aggregate builder 170 accesses the metadata 160 to build the aggregates. Often, the aggregate building is done at night.

After aggregates are built, the querying and reporting process 206 can be performed. The querying and reporting process 206 can be performed anytime after the creation of the datamart 150. Importantly, when an aggregate is created, the appropriate operation for that aggregate is used. For example, revenue elements are added to produce an aggregate, while daily account balances are averaged to produce an aggregate.

At block 277, the consultant defines the query mechanism schema for the system 100. In particular, the consultant defines the query/reporting information 169 and the measurement information 168. These two pieces of metadata 160 allow the system 100 to report meaningfully consistent information to users. Also, the consultant is not burdened with having to hand create the possible reports.

At block 280, a query is generated. In one embodiment of the invention the query is generated at the query/reporting program 104. In other embodiments, the query can be generated at the user computer 180 through the HTTP, web server 186, Java coupling to the query/reporting program 104. What is important here is that some query is generated that can be used to access the datamart 150. Importantly because the schema definitions 161 are available to the query and reporting program 104, the user can be presented with forms from which a query can be easily and automatically generated.

At block 290, the answer set (the results) is created by the datamart 150. This answer set is then propagated back through the query/reporting program 104, and ultimately to the user computer 180. The results are formatted according to the query/reporting information 169.

Top Level Metadata Schema

As noted in the background, multi-dimensional datamarts use star schemas. The system 100 uses star schemas in a larger organization that allows for the sharing of dimension tables by sets of similar facts. This larger organization is called a constellation. Figure 3 illustrates a schema for the schema definitions tables that support constellations. (The schema of Figure 3 is labeled the schema for schema definitions 300.) That is, Figure 3 illustrates a schema used in the system 100 to define schemas for the datamart. Figure 3 also illustrates some of the aggregate information 167 schema.

The following describes the meaning of the various graphical elements in Figure 3 through Figure 5. Each box in the figure represents a table having one or more attributes. A first table having a diamond graphic extending to second table (with a dot on the end) indicates that that

second table has a foreign key pointing to the first table. This can be thought of as a parent child relationship.

It is important to remember that Figure 3 through Figure 5 illustrate the schema of the system used to generate and run the datamart 150. Rows in these tables define the schema for use in the datamart 150. From these rows, create table, table query, etc., commands are created. These commands are used to create the tables in the datamart 150 and to access that datamart.

Also, as mentioned previously, the datamart 150 is well-formed because, among other reasons, the system 100 automatically includes additional columns in the table created in the datamart 150. For example, source system key, foreign key, and time and date columns are automatically added (where appropriate). The rest of the elements of the system can then rely on the existence of these columns. This prevents, for example, the creation of an inconsistent schema where only some of the tables include date and time information.

The following first lists all of the elements in Figure 3 and then describes those elements and their relationships.

Top Level Metadata List

Figure 3 includes the following elements: a constellation 302, a fact table 304, a dimension base 306, a semantic instance 308, a fact column 310, a fact aggregate operator 312, a fact column number 314, a fact dimension cleansing 316, a dimension role 320, a degenerative number 322, a dimension role number 324, a dimension node 326, a dimension column 329, a dimension column number 321, a cleanse type 323, a cleanse map definition 327, a cleanse map 325, a physical type 330, a transaction string 332, a metacolumn 334, an actual table type 336, a

dimension base type 328, a special dimension base 391, an aggregate key operator 392, an aggregate dimension type 393, an aggregate dimension 344, an aggregate group 342, an aggregate fact 340, a aggregate dimension set 372, a dimension column set 370, a dimension column set definition 374, a fact index 380, a fact index definition 384, and a fact index number 382.

Top Level Metadata Descriptions

It is important to remember that the tables in Figure 3 are only used to define the schema in the datamart 150. Thus, a fact table 304 in Figure 3 is not the actual fact table in the datamart 150, but the definition of that fact table. Each row in a table corresponds to an instance of that table.

The constellation 302 defines the organization of the schema in the datamart 150. It is the top level table in the schema definition.

Fact Related Tables

The fact table 304 defines the metadata 160 table describing all of the fact tables within a given constellation 302. The attributes of the fact table 304 include a build aggregates flag, a cleanse flag, a constellation key, a description, a fact table key, a fact table name, and a truncate stage flag. Each attribute corresponds to a column in the fact table 304. The build aggregate flag indicates whether or not to build aggregates for a particular fact on the next execution of the aggregate builder 170. The cleanse flag is a flag that is used in many of the tables to obliterate the actual measures within a table in the datamart 150 (particularly useful in demonstrations of the system 100 where sensitive data would otherwise be revealed). The constellation key points

to the parent constellation 302 for a given fact table 304. The fact table name is the name of the fact table used in constructing the corresponding physical table names in the datamart 150. The truncate stage flag is used to indicate whether or not to truncate the fact staging table on the next extraction.

5 The fact column 310 lists all of the fact attributes within a single fact table 304. The fact column 310 includes a cleanse flag, a description, a fact aggregate operator, a fact column key, a fact column name, a fact column number, a fact table key, and a physical type. The fact aggregate operator is an SQL operator used to aggregate this fact column in the datamart 150. The fact column key is the primary key for the fact column. The fact column name is the physical name of the fact column. The fact column number counts and orders the number of columns in the fact table. The fact table key points to the fact table to which the corresponding fact column belongs. The fact table key points to the fact table to which the fact column belongs. The physical type is the database type for the fact column. This type is a logical type and provides for independence of implementation of the datamart 150 from the underlying database used.

The fact column number 314 and the fact aggregate operator 312 are used by the fact column 310. These have already been described in the context of the fact column 310.

The fact dimension cleanse table 316 has rows that indicate the dimension foreign keys in a fact that should be cleansed. The fact dimension cleanse table 316 includes a dimension role key, a fact dimension cleanse key, and a fact table key. The dimension role key indicates that this dimension role 320 is part of the "group by" set for cleansing a fact table without distorting

trends in the data. The fact dimension cleanse key is the primary key for the fact dimension
cleanse table 316. The fact table key is the fact table having its cleansing properties.

Dimension Related Tables

The dimension base 306 is the metadata 160 describing all the dimension tables that can be
5 used in a given constellation 302. These dimension bases can then be used in multiple
constellations. The dimension base 306 includes the following attributes: an aggregate key
operator, a cleanse flag, a description, a dimension base key, dimension base name, a dimension
base type, and a truncate stage flag. The aggregate key operator is an SQL operator used by the
aggregate builder 170 to build aggregates from a dimension. The cleanse flag and description act
10 similarly to those attributes in other tables. The dimension base key is the primary key for the
dimension base 306. The dimension base name is the name of the base dimension used in
constructing real tables in the datamart 150. The dimension base type indicates the type of a
dimension base (either default or special (special includes "date" and "transaction type," which
are used by the system 100). The truncate stage flag operates in the manner similar to other
15 truncate stage flags.

The dimension column 329 defines the list of dimension attributes that are valid for a single
base dimension 306 and inherited by a dimension usage. The dimension column 329 includes a
cleanse label, a cleanse map key, a cleanse type, a description, a dimension base key, a
dimension column key, a dimension column name, a dimension column number, a dimension
20 number key, grouped by field, a physical type, a primary key, a time navigation field, and a
default value. The cleanse label is a label presented to users after this column has been cleansed.

The cleanse map key is for use when cleansing using value mapping. The cleanse map key indicates the mapping group to use. The cleanse type is the method for cleansing the dimension column 329. The description is for documenting the dimension column 329. The dimension base key is the numbered base in which the column resides. The dimension column key is the primary key for the dimension column 329. The dimension column name is the physical name of the column. The dimension column number is the count of the dimension columns (to prevent too many from being created in the datamart 150). The dimension node key is the aggregate hierarchy group in which the column resides. The “group by” field is used for special dimensions to indicate whether or not this column needs to be “grouped by” during the processing by the aggregate builder 170. The physical type is a logical database type for this dimension column 329. The primary key is used in special dimensions to indicate whether or not this column is the primary key. The time navigation field is for the date special dimension to indicate whether or not time navigation should use this field. The default value is the default value for the dimension column.

The dimension column number 321 is a look up table for the valid number of dimension columns that can be created. The dimension column number counts the number of dimension columns to make sure there are not too many being defined by the consultant.

The dimension role 320 is a metadata 160 table that describes all of the dimension tables used in a constellation 302. The dimension role 320 includes a constellation key, a degenerative number, a description, a dimension base key, a dimension role key, a dimension role name, and a dimension role number. The constellation key points to the constellation 302 in which the

dimension role 320 resides. The degenerative number defines the order of degenerate columns within fact tables in a constellation. The description is a documentation field for describing a dimension role. The dimension base key is the dimension base that this dimension role refers to. The dimension role key is the primary key for the dimension role 320. The dimension role name
5 is the name of the dimension role and is used when constructing the foreign keys in the fact tables in the datamart 150. The dimension role number defines the order of the dimension roles within a constellation. That is, a constellation may have multiple dimension roles and the dimension role number allows for an ordering of those dimension roles.

The dimension node 326 is a table used for defining and grouping hierarchical dimension
10 attributes that are used by the aggregate builder 170. The dimension node includes a dimension base key, a dimension node key, a node name, a node number, and a parent node number. The dimension base key points to the dimension base being defined. The dimension node key is the primary key for the dimension node. The node name is the logical name for the aggregate
15 hierarchy group being defined. The node number is the logical number for the aggregate hierarchy group being defined. The parent node number is a logical number for the parent of the aggregate hierarchy group being defined.

The degenerative number 322 and the dimension role number are defined as described in the dimension role 320.

The dimension base type 328 is defined as described in relation to the dimension base 306.

Semantic Instance Table

The semantic instance 308 is a single record that represents the manner in which a fact or dimension table is extracted from staging tables, manipulated, and then used to populate the corresponding table in the datamart 150. The semantic instance 308 includes an extraction node key, dimension base key, a fact table key, a semantic instance key, and a semantic type key. The extraction node key points to the extraction node that a particular semantic instance belongs to. The dimension base key is the dimension base table owning this semantic instance. The fact table key points to the fact table owning this semantic instance. Only one of the dimension base key and the fact table key is filled in for a semantic instance 308 because the semantic instance can only be applied to one or the other. The semantic instance key is a primary key for the semantic instance 308. The semantic type key is the indicator of the type of transformation necessary to construct this type of semantic instance in the datamart 150.

Aggregate Related Tables

The aggregate builder 170 is a program that uses the aggregate tables and the schema definitions 161 to build aggregates. Often this will be done on a nightly basis.

The aggregate group 342 defines a set of aggregates to be built for a constellation. An aggregate group 342 will cause a combinatorial creation of many aggregate tables in the datamart 150. The consultant defines for which dimensions aggregates are to be built (e.g., the consultant will define that one, none, all, etc. columns of a dimension are to be aggregated on in an aggregate group). The aggregate filtering done by the query and reporting program 104 will select the most appropriate aggregates for a given query.

The aggregate group 342 includes an aggregate group key, an aggregate group name, a constellation key, a default flag, a description, and an enabled field. The aggregate group key is the primary key for the aggregate group 342. The aggregate group name is the logical name of this aggregate group. A constellation key points to the constellation in which this aggregate group resides. The default flag indicates whether or not this group is the default group within a constellation. Default groups have facts and dimensions automatically added to them. They can also not be deleted. The description contains the documentation for this aggregate group. The enabled field indicates whether or not the aggregate builder 170 will actually build this group.

The aggregate fact table 340 tracks the membership of a fact within an aggregate group. The aggregate fact table 340 includes an aggregate fact key, an aggregate group key, and a fact table key. The aggregate fact key is the primary key for the aggregate fact 340. The aggregate group key is the aggregate group being defined by the aggregate fact. The fact table key points to the fact table that is being made a member of the group.

The aggregate dimension 344 indicates the membership of a dimension within a constellation in an aggregate group. The aggregate dimension 344 includes an aggregate dimension key, an aggregate dimension type, an aggregate group key, a dimension role key, and a special dimension base key. The aggregate dimension key is the primary key for the aggregate dimension 344. The aggregate dimension type indicates the manner in which this dimension (special or role) will be included in the aggregate group. The aggregate group key indicates the aggregate group being defined. The dimension role key points to the dimension role being

included. It is possible that this key is null. The special dimension base key indicates the special dimension being included. The special dimension base key can also be null.

The aggregate key operator 392 is defined as described in the dimension base 306.

The fact aggregate operator 312 is a look up table of valid fact aggregation operations. Each operator is an SQL operator used to aggregate a fact column.

A special dimension base 391 provides details about special built-in dimensions in the system 100. The special dimension base includes an "always include an aggregate" field, a default aggregate dimension type, a dimension base key, a list order in fact, a physical type of key, an index flag, and a special dimension base key. The "always include an aggregate" field indicates whether or not this dimension table must always be included in all aggregates. The default aggregate dimension type is the default manner in which this dimension is included in aggregate groups. The dimension base key is the one to one relationship to a dimension base. The list order in fact is the order in fact tables that the foreign key to this table will be listed. The physical type of key is the logical database type that foreign keys in the fact tables that point to this special dimension will be. The index flag is used in indexing. The special dimension base key is the primary key for the special dimension base 391.

Data Store Related Tables

The physical type 330 defines a look up table of logical data types that are relational database management system (RDBMS) independent. The physical type 330 is a logical data type that works across various source systems storage types. The physical type 330 includes a database physical type, a default value, and a special type.

The translation string 332 defines a list of strings that are translated for different RDBMS's. The translation string is a logical string that can be converted to specific strings for different storage types. Each storage type would correspond to a different source system 110.

Cleansing Related Tables.

5 The cleanse type 323 is a look up table to indicate how to cleanse a dimension column.

The cleanse map 325 is a mapping table for mapping real names to cleanse names. The cleanse map 325 includes a cleanse map key, which is the primary key, and a cleanse map name, which is the name of a set of mapping pairs for the purpose of scrambling data.

The cleanse map definition 327 defines the details of what should be mapped to which fields.

10 The cleanse map definition 327 includes cleanse map character ID, a cleanse integer ID, a cleanse map definition key, a cleanse map key, and a cleanse value. The cleanse character ID is a character value for indexing into this mapping group. The cleanse integer ID is a numeric value for indexing into this mapping group. The cleanse map definition key is the primary key for the cleanse map definition. The cleanse map key is the mapping set to which this particular
15 cleanse map definition belongs. And the cleanse map value is the translation value after the mapping.

Additional Tables

The metacolumn 334 is a column that occurs by default in tables in the datamart 150. The metacolumn 334 includes an actual table type, a list order, a metacolumn key, a metacolumn
20 name, and a physical type. The actual table type indicates the type of physical table in which this special column should appear. The list order is the order this column occurs in tables of the

appropriate type. The metacolumn key is the primary key. The metacolumn name is the physical name of the column when it is used. The physical type is the logical data type for this column.

The actual table type 336 is a look up table for actual table types. Actual table types can be fact, dimension stage, fact stage, dimension map, or dimension.

The aggregate group 342, the aggregate fact 340, the aggregate dimension set 372, the dimension column set 370, the dimension column set definition 374, the fact index 380, the fact index definition 384, and the fact index number 382 are for future use and are therefore optional. Each of these tables provides greater flexibility when defining the metadata 160, improves the performance of the system 100, or may otherwise enhances the system 100.

Top Level Metadata Use

Now that all of the elements in Figure 3 have been listed and described, their relationships and workings are now described.

It is important to note that many of the tables in Figure 3 are actually used in providing layers of abstraction to allow for the reuse of information and non-abstract tables. Therefore, a consultant will often only deal with only some of the tables in the Figure 3. For the purposes of describing how the metadata 160 can be used to define a schema for the datamart 150, these grouping and levels of abstraction tables will be described where appropriate.

Generally, a consultant will create a new datamart 150 by defining instances of the dimension bases 306, and constellations 302. Each instance corresponds to a row in the dimensions bases 306 table or the constellation 302 table. The constellation instances are defined by defining

aggregates, dimensions, facts, measures, and ticksheets. The following describes the definition of a schema using the metadata 160. This corresponds to block 210 of Figure 2.

Beginning with the facts in a constellation, the consultant defines a fact table 304 row that will define the hub table in a star schema supported by the constellation. Again, it is important to remember that the fact tables in Figure 3 are for definitional purposes, and are not the real fact tables in the datamart 150. A row in the fact column 310 holds the details of what columns will be created for place holders of actual values in a corresponding fact table. Thus, for each fact, the consultant defines the various fact columns.

Once the facts have been defined, the consultant can then define the dimensions of the constellation.

Remember that the dimension base 306 holds the information to define the actual dimensions of the tables in the datamart 150. The dimension role 320 allows for the reuse of the dimension base tables. Thus, different dimension roles can refer to the same dimension base. This provides an important feature of some embodiments of the invention where the same dimension bases can be used in multiple constellations or within the same constellation. The dimension columns 329 define the columns on which queries can be performed in the datamart 150. The dimension node table 326 helps relate the dimension columns 329. Thus, the consultant will have defined the basic schema for the datamart 150.

The aggregate group 342 defines how particular facts or dimensions are to be aggregated by the aggregate builder 170. These aggregated facts provide much faster queries in the datamart 150.

The cleansing map tables are for scrambling the data in the datamart 150 for presentations to people who want to see the functionality of the system 100, without having to reveal the actual data in the datamart 150.

The special dimensions are the transaction type table and date values that are included in every fact table. Because this is included in every fact table, the system 100 can rely on the existence of the transaction type during the various stages of datamart 150 creation, modification, querying, and the like.

Thus, the elements of Figure 3 can be used to allow the consultant to define the schema definitions 161 for creating the tables in the datamart 150.

Extraction Metadata

The following describes the metadata 160 used in the extraction process 204. This metadata, represented as extraction schema 400, is shown in Figure 4. The extraction process focuses around the job and connector tables. In general, these tables define the various steps in extracting the source system data into the staging tables 130 and performing the desired semantic conversions on that data.

Extraction Metadata List

Figure 4 includes the following elements: a job 402, a job step 404, a system call 405, a connector 406, a connector time stamp 407, a connector step 408, a connector column latch 409, and an extraction group 411, an extraction note 410, an SQL statement 420, and error handling type 413, an external table 422, an external column 424, the physical type 330, the fact table 304,

a debug level 415, the semantic instance 308, a semantic type 430, a dimension semantic type 432, a fact semantic type 434, the actual table type 336, a semantic type definition 436, an adaptive template 438, and adaptive template block 439, the dimension base 306, a job log 401, a connector store role 448, a store role 446, a statement type enabled 428, a store role allow 444, a data store 440, a source system 442, a file store 441, and Oracle store 454, a store version 452, and SQL server store 456, and ODBC store 458, and a store type 450.

Extraction Metadata Descriptions

Job Related Tables

The job 402 is a top level object for controlling the work flow during the extraction and loading process 204. The job 402 includes a check databases field, a check tables field, a description, a label, an initial load flag, a job key, a job name, a log file width, a mail to on error, a mail to on success, and a truncate flag. The check databases field indicates whether or not an attempt should be made to log into all the data stores before executing the job. The check tables flag indicates whether or not to check for the existence of all the tables in the datamart before executing the job. The description is for documenting the job (usually done by the consultant). The enabled flag indicates whether or not a particular job can be run. The initial load flag indicates whether or not to ignore all previous time stamped constraints when running a particular job. The job key is the primary key for the job table 402. The job name is the internal name of the job. The log file width indicates how many characters wide to make rows in the log file output. The mail to on error, and the mail to on success indicate where E-mail messages

should be sent after failure or success of the particular job. The truncate flag indicates whether or not to truncate any tables when running a job.

The job log 401 is the locations where the running job is logged. That is, the location of the output that will be provided to the consultant indicating what occurred during the extraction (e.g., what errors occurred). The job log 401 includes a data store key, a job key, a job log key, and a job store role. The data store key indicates the data store having a role defined within the job. The job key is a reference to the particular job, the job log key is the primary key for the job log 401. The job store role is the role being assigned to that particular job log 401. An example job store role is "<working directory>," indicating the path to the working director where job log files are store.

The job step table 404 includes the detailed steps that make up a job. This includes connectors and system calls. The job step table 404 includes the following attributes: a connector key, a description, an enabled flag, a job key, a job step key, a list order, a phase, a job step type, and a system call key. The connector key indicates the connector being included in any particular job step. The connector key can be null. The description is for documenting the job step. The enabled flag indicates whether or not a particular step is enabled. The job key points to the job being defined by the job step 404. The job step key is the primary key. The list order indicates the order of a particular job step. The phase also indicates the order of a particular step. By supporting both list order and phase, alternative embodiments of the invention can support parallel extraction. Steps in the same phase can then be executed simultaneously. The job step type indicates the type of the job step (which are defined in the job

step type table 481). The system call key points to a system call included in a particular job step.

The system call key can be null.

The system call 405 is a table including external OS system calls. These external system calls can be used to perform any number of external system functions. The system call 405

5 includes the system call key, a command string, a description, a name, and an on-error type. The system call key is the primary key for the system call. The command string is the actual operating system command to be run as a result of the system call. The description is for documenting the system call. The name is the logical name of the system call being defined. The on error type is an indicator to point to what to do if the system call fails.

10 Connector Related Tables

The connector 406 defines a name and a description. The connector 406 is a grouping mechanism for extraction statements and a specification for input and output data stores. The description is used for documenting the connector. The name is the logical name of the connector. The connector 406 represents an ordered collection of connector steps 408.

15 The connector step 408 defines steps within a connector. The connector step table 408 includes the following attributes: a connector key, a connector step key, an enabled flag, an extraction node key, a list order, and a phase. The connector key points to the connector being defined. The connector step key is the primary key. The enabled flag indicates whether a particular connector step is enabled. The extraction node key points to an extraction node that is, 20 the extraction group of statements and semantics that make up this connector step. The list order is the order of steps in the connector. The phase is also the order of the steps in the connector.

extraction succeeds. The last maximum value is the maximum value that was extracted during the last run of the extraction.

The connector store role 448 defines the usage of a data store for a particular connector. It indicates whether the data store is input or output. The connector store role points to the connector and the data store. The connector store role also indicates the type of storage usage being defined for this connector (input or output).

Extraction Group Related Tables

The extraction group 411 defines a group of extraction steps. The extraction group 411 includes a description, and a name of the set of extraction steps.

The extraction node 410 is a single node, or step, in the extraction tree. The extraction tree defines the order of extraction steps. An extraction node includes an extraction node type, a debug level, a debug level row, an enabled flag, an extraction group key, an extraction node key, a list order, and on error type, a parent extraction node key, and a phase. The extraction node type defines the type of extraction node (as defined in the extraction node type table 491). This relationship is important and allows for the conversion of data in the staging tables for use in the datamart 150. The debug level indicates how to debug a particular step during execution. The debug level row indicates which row to start debugging at for SQL statements. The enabled flag indicates whether or not to execute a particular SQL statement associated with the extraction node. The extraction group key points to, if not null, the name of the group. The extraction node key is a primary key. The list order and the phase define the order of the corresponding step within an extraction node's parent. The on error type indicates what to do if there is an error

during the execution of the step associated with the extraction node. The parent extraction node key points to the parent extraction node of the present extraction node.

SQL Statement Related Tables

5 The SQL statement 420 defines a single step in an extraction run. A row in the SQL statement 420 table represents an SQL statement. The columns in the SQL statements match those in the corresponding dimension base definition or fact table definition. In one embodiment the consultant supplies the SQL source in SQL statements.

10 The SQL statement 420 includes an extraction node key, a description, a dimension base key, and execute against input flag, an external table key, a fact table key, SQL source and SQL statement key, and an SQL statement name. The extraction node key points to the extraction node associated with a particular SQL statement 420. The description is for documenting the SQL. The dimension base key points to the dimension base for the corresponding SQL statement. The execution against input flag indicates whether or not to execute this SQL statement against the source or destination data store of the connector that is calling this SQL statement. The external table key points to the external table, if any, being extracted into. The fact table key points to the fact table, if any being extracted into. The SQL source is the actual SQL source to be executed during the SQL statement execution. The SQL statement key is the primary key. The SQL statement name is the logical name for an extraction SQL statement.

15 The statement type enabled 428 defines which RDBMS types use this extraction statement.

20 The statement type enabled 428 includes the following attributes. An SQL statement key, a statement type enabled key, a store role name, and a store type. The SQL statement key points to

the SQL statement. The statement type enabled key is the primary key. The store role name is the role name for which this SQL statement should be used. The store type is the store type for which this statement should be executed.

The external table 422 defines the user defined destination table for use during a multiphase
5 extraction. This user defined destination table can be used to temporarily store data during an extraction. The external table 422 defines the physical name of the external table and whether or not to truncate this external table on the next job run.

The external column 424 defines a column in a user defined extraction table. The external
column 424 includes the attributes for documenting a particular external column, and the column
10 name in the external table. A pointer to the external table, a list order of appearance in the external table, and a physical type of the logical database for this column are included in the external column 424.

Error Handling

The error handling type 413 is a look up table to define how to respond to a particular error.

15 An error handling type can be a default action to take when an error occurs.

The debug level 415 defines ways in which extraction steps can be debugged. The debug level includes a logical name for users to pick a debugging behavior. The default level also includes a default flag.

Data Semantic Related Tables

20 The semantic type 430 defines a set of predetermined semantic types for use in defining a schema. The semantic type includes a logical name for a particular transformation. Associated

with the semantic type are a dimension semantic type 432 and a fact semantic type 434. The dimension semantic type table 432 defines the ways in which dimension data in the staging tables 130 can be extracted and put into the datamart 150. Similarly, the fact semantic type defines the ways in which the information in the staging tables 130 can be put into the fact tables of the datamart 150. Both the fact semantic type 434 and the dimension semantic type 432 include pointers to an actual table type and are used to subset the full list of semantic types.

Each semantic type 430 is made up of a semantic type definition. The semantic type definition table 436 defines the set of adaptive templates used in any given semantic type. The semantic type definition 436 includes the semantic type key that points to the semantic type 430 for a particular semantic type definition. The semantic type definition also points to the adaptive template 438 used. The semantic type definition also includes a list order for the ordering of the adaptive templates.

The adaptive template 438 is a semantic transformation template (e.g., an SQL program) that is used in the extraction of the data in the staging tables 130 to turn all source data into transactional data. The adaptive template 438 includes attributes indicating a logical name for an adaptive program used within semantic transformations.

The adaptive template block 439 defines the individual pseudo-SQL statements that make up a template. The adaptive template block 439 has the following attributes: the adaptive template block key, an adaptive template key, a block name, a list order, an on error type, and SQL source. The adaptive template block key is the primary key. The adaptive template points to the adaptive template to which this block belongs. The block name is the internal logical name for this block

of pseudo-SQL source code. The list order is the order of this block within the template. The on error type indicates what to do if this block causes an SQL error when executed. The SQL source includes a template of pseudo SQL source code. This template is described in greater detail below.

Data Store Related Tables

The store type 450 defines the types of RDBMS's supported by the system 100.

The data store 440 defines a logical data source, or sink, used during the extraction. The data store 440 includes the following attributes: the data store key, a data store flag, a description, a name, a source system key, and a store type. The data store key is the primary key. The datamart flag indicates whether or not this data store is the special datamart store. Since the datamart 150 and the metadata 160 can reside in the same database or different, the data mark helps resolve the location of the datamart 150. The description is for documentation of the particular data store. The name is the logical name of the data store. The source system key points to the source system identifier to which this data store belongs. This allows live, and backup, source systems to share the same identifier. The store type indicates the store type of this data store.

The source system 442 is a logical identifier for a source system 110 from which data can be pulled. This allows two physical databases to act as one master database and one backup, for example. The source system 442 includes a description attribute, a source system key and a source system name. The description is for documentation to describe the source system. The source system key is a primary key for this table. This number also becomes identified source

system field in the staging tables 130 being filled. The source system names is a logical name for a source system 110 from which the system 100 is pulling data.

The following store tables are subtypes of the data store table 440. They address specific data stores.

5 The file store table 441 defines the files in which data can be stored. The file store 441 defines a directory and file name for each file.

10 The Oracle store 454 defines information about particular Oracle databases. The Oracle store 454 includes the following attributes: a data store key, an instance name, an Oracle store key, a password, an SQL network name, a user name, and a version. The data store key is a one to one relationship key to the data store being defined. The Oracle store key is the primary key. The password and user name are used to access a specific Oracle system. The version number is the Oracle vendor version number. The SQL network name is the SQL net instance name. The store version is a version of the store type (the database vendor's version for example) that the system recognizes. The store version has a pointer to the store type being defined and also includes a version number attribute.

15 The SQL server store table 456 defines details about an SQL server system. The SQL server store includes the following attributes: a data store key, a database name, a password, a server, and SQL server store key, a user name, and a version. The data store key is a one to one relationship to a data store entry. The database name is an SQL server database name (\$\$DEFAULT means the database in which this role resides). The password is the SQL server password. \$\$ DEFAULT again means the password currently logged into to read this data. The

server is the SQL server name. The SQL server store key is the primary key. The user name is the SQL server user name. The version is the vendor's version number of this SQL server. \$\$DEFAULT means use the default value for the current database being used. For example, the database name means the database in which this role resides.

5 Extraction Metadata Use

The following describes the tables of Figure 4 in the context of the extraction process 204. The job, the job step, and the connector, group extraction steps for extracting information from the source systems 110 and cause that information to be placed in the datamart 150. This organization allows for a very flexible extraction process. For example, where a two phase extraction is required, one connector could be used to extract the information from the source system 110, while a second connector could then be used to take this extracted data from an external table.

The job defines the order of the execution of the connectors. The job also allows for the running of an external program, such as system call, between connector executions. Thus each, job step in a job can be a system call or a connector.

The following is an example illustrating the organization of job. Assume that a consultant wants to extract information from a source system that provided a raw set of home addresses. A system call could be run as part of a job step. The system call would determine the zip codes associated with those addresses. The zip codes could then be included in the datamart 150.

20 The relationship between the connector 406, the connector step 408, and the extraction node 410 is that the connector 406 allows for the reuse of extraction nodes in multiple connectors

(through the connector step). This relationship is particularly important where similar connectors are created. For example, assume that a consultant wants to create a connector that runs some steps Monday through Friday and different steps on Saturday and Sunday. Most of the steps in the connector will be common, however some will be different. Through the use of the connector step, the consultant can reuse many of these connector steps in each connector.

As noted before an extraction node can be a leaf or a grouping mechanism for extraction. It can correspond to an SQL statement extraction step or a semantic conversion step. During the definition of the schema, the consultant defines the specific SQL statements for extraction and the specific semantic instances for the facts and dimensions.

An SQL statement is a single step in an extraction run that represents a data push or a data pull. The SQL source code dictates the action for a given extraction node. After the SQL statements are run, the staging tables 130 are ready. The semantic conversion of the data in the staging tables 130 can occur.

The semantic instance represents the use of a single generic template on one fact or dimension table. The semantic type associated with the semantic instance is one of a number of pre-defined recognized data meanings within the system 100 (e.g., an "order"). The semantic types correspond to programs for converting the data in the staging tables 130 into data for use in the datamart 150. An example of a semantic type is a "slowly changing dimension" type.

The semantics types, as mention previously, are made up of a series of templates. These templates include tokens that can be replaced with information from the corresponding dimension base or fact table. An example of an adaptive template is one that would be used in

re-indexing of a fact table. This could be used as the last step in the semantic transformation of facts. The re-indexing will help speed the operation of the datamart 150. Importantly, this same indexing is performed for each fact table. No matter which semantic type is chosen for a given fact table, the same indexing is performed. Thus, this adaptive template can be used in each semantic type through its semantic type definition.

The following describes the slowly changing dimension semantic type. (See Appendix A.) In this semantic type are an insert dimension and an index dimension adaptive template. Each adaptive template has a corresponding set of pseudo-SQL statements. During the semantic template conversion 140 this pseudo-SQL will be transformed into real SQL source code. This is done by converting the pseudo-SQL tokens into actual dimension column names, etc. (the column names and table names are derived from the schema definitions 161).

Thus, during the extraction, the extraction node associated with a particular semantic type instance is processed. This causes the adaptive template blocks associated with the semantic instance to be processed. The dimension information associated with that semantic instance, or the fact table information associated with the semantic instances, can then be used to replace the tokens with the specific information associated with that dimension or that fact.

Some embodiments of the invention correspond to only one or more semantic templates and a computer readable media, a computer, an electromagnetic waveform, or the like.

Further Discussion of Templates

The following describes the pre-parsed template and the post-parsed results in greater detail. Each token begins with a \$\$\$. In the example template, for the slowly changing dimension

semantic type, a number of tokens begin with \$\$ DIM KEY. Similarly, tokens appear that begin with \$\$FSTGTBL[]. In the post-parsed template, the dimension key tokens have been changed to cost center keys, account key, subaccount keys, etc. Note any tokens, and their surrounding text, that are not replaced are removed from the post-parsed text. If more tokens need to be replaced then are available in the template, then an error flag will be set. In other embodiments of the invention, the templates are dynamically generated according to the number of columns defined in the schema definition 161. In other embodiments, templates are not used but the "post-parsed SQL" results are dynamically generated from the schema definitions 161 and the semantic instance types.

In this example, the net price corresponds to a fact column in a fact table. This indicates that the table entitled SSA in the post-parsed example includes one fact called net price.

In some embodiments, the pre-parsed templates include additional tokens to deal with specific data stores. For example, the "select into" statement is a token in the pre-parsed version. This compensates for whether the data store is in Oracle database or an SQL server.

Another feature of the pre-parsed language is that "--#begin#" is used to break the pre-parsed version into adaptive template blocks.

Examples

Appendix A illustrates semantic types that may be supported and their corresponding adaptive template names. For example, the Pipelined semantic type is made up of, in this order, the map_keys the pipe_state and the index_fact adaptive templates. The example pre-parsed and post parsed SQL adaptive templates are then provided.

As mentioned previously, the use of the semantic types significantly reduces the amount of work needed to implement the datamart 150. By selecting a semantic type for a particular fact table or dimension table, the consultant automatically selects the corresponding pre-parsed SQL adaptive templates. The selected adaptive templates are then automatically converted into post-parsed SQL statements that include the schema specific information for the datamart 150. Additionally, these post-parsed SQL statements include the SQL for converting the data in the staging tables 130 into data that can be used in the datamart 150 tables.

Additional Templates

Two types of templates not described in Appendix A are "team" templates and "denormalized" templates.

The team template is used to properly populate an "associative" dimension table. Such a table is used whenever there is a one-to-many relationship between an individual fact row and a dimension. For example, if multiple salespeople can split the credit for an order, one needs some way to represent this situation in the datamart. In a star schema, one normally associates a tuple of dimension values with a fact row (e.g. product, customer, salesrep dimensions for the fact row containing price, quantity etc.). Since there is only a single salesrep_key, one could normally have only one salesrep associated with this transaction. There are two solutions. One is to introduce multiple fact rows for a transaction involving one to many relationships. If there were three salesreps on a specific order, there would be three fact rows for this order stored in the database. This has the disadvantage of multiplying the data size by a factor of three and slows queries. Also queries that are concerned with the total number of transactions become more

difficult to process since duplicate rows, due to the multiplication by the number of salesreps, must be eliminated.

Another solution is to introduce an associative table between the actual salesrep dimension table and the fact table. Conceptually, the associative table contains "teams" of salespeople. If salesreps A, B and C often sell products together, they will be associated with a unique team key. The team key will be stored in each fact row for orders sold by the A, B, C team. The associative table will associate the team key with the three rows for A, B and C in the salesrep table. The associative table will have 3 rows representing this team (A-key, team1-key), (B-key, team1-key) and (C-key, team1-key). If the team of A, B, D and Q also sold products together, the associative table would have four additional rows (A-key, team2-key), (B-key, team2-key), (D-key, team2-key), (Q-key, team2-key). The team template scans the staging table used to load the fact table and generates the appropriate rows for entry into the associative table, only for those teams THAT ACTUALLY OCCUR in the fact rows being loaded.

Also, if a team is already present in the associative table it will be reused.

In real world situations, the number of teams that actually occur is much smaller than the total space of all possible teams.

Note that this team template can be used wherever there is a one to many relationship between fact and dimension rows.

Another example is in a Sales Force Automation system where the fact rows correspond to a sales "opportunity".

An opportunity may be associated with the dimensions of Sales Lead Source, Product and Customer Contact. All of these may be one to many relations, amenable to the "team" concept.

As mentioned above, the other type of template is the denormalized data template. This is a variant of the

5 "Team" template where instead of introducing the extra associative table between the dimension and fact tables, the dimension table is a combination of the associative table and the actual dimension table. This effectively flattens the data. In the above example the dimension table would contain rows like ("Greg Walsh", A-key, team1-key),

10 ("Craig", B-key, team1-key), ("Ben", C-key" team1-key), ("Greg Walsh", A-key", team2-key) etc. Greg Walsh is a member of both teams 1 and 2 and his name (and

other attributes) rather than just his key (A-key) is stored twice. Used judiciously this can result in faster queries than the associative table case but results in duplicate data being stored.

15 The population of both the denormalized team dimension and the associative table are difficult to code properly. This is especially true if this is done incrementally (e.g., on nightly extracts) and if you want to be independent of team order (e.g. A, B, C) is the same as (A, C, B). Thus, allowing the consultant to simply select this data semantic provides a significant improvement over previous systems.

Runtime Metadata

20 Figure 5 illustrates the schema for the runtime environment within the system 100. The runtime schema 500 represents the schema description for the schema of the running datamart 150. That is, when the datamart 150 is created or modified, the schema definition is propagated

into the runtime schema 500. Thus, the runtime schema 500 allows for the datamart 150 to be changed without having to rebuild all the tables and repopulate all of those tables. Additionally, the runtime metadata 500 provides the support for aggregate navigation. Aggregate navigation involves determining which aggregate to use in response to a query. Schema modification and aggregate navigation will now be explained in greater detail.

The schema modification involves comparing the changed schema definition with the present schema definition. As will be seen below, an actual table 502 keeps track of all of the dimension tables and the fact tables in the datamart 150. When a change is made to the schema definition, a comparison is made between the old definition and the new definition. The difference between these definitions defines the set of tables, columns, and rows that need to be added, deleted or modified, in some way. Importantly, the modifications can often be made without losing any data in the datamart 150.

The aggregate navigation process determines which aggregate most closely suits a particular query. The runtime metadata 160 keeps track of the aggregates available in the datamart 150. The query and reporting program 104 initiates a view of the runtime metadata 500 (in particular, the tables holding the aggregate tables definitions). The view results indicate which aggregates are available to answer the particular query. The view results are further examined to determine the best aggregate to use (the one that most closely corresponds to the query).

Importantly, the query machinery does not need to be aware of aggregates to be able to take advantage of them. Aggregates are simply presented to the query machinery as a solution to a query.

Additionally, aggregates can use other aggregates to build themselves. This is because the schema definition can be used to determine the relationship between aggregates.

Runtime Metadata List

The runtime schema 500 includes the following elements: an actual table 502, an actual column 504, a fact aggregate table 512, a fact aggregate dimension 514, a dimension base aggregate 516, a dimension base aggregate column 518, a datamart letter 510, the dimension base 306, the fact table 304, the external table 422, an actual column 504, a physical type definition 530, an actual table type 336, an actual column type 540, the physical type 330, a database physical type 595, the translation string 332, a translation actual 539, a store type 450, a date 560, a business process 570, an adaptive template profile 580, and a transaction type 590.

Runtime Metadata Descriptions

The actual table 502 corresponds to metadata 160 that describe which dimension base and fact tables “actually” exist in the datamart 150.

The actual table 502 includes the following attributes: an actual table key, an actual table name, an actual table type, a dimension base key, an external table key, a fact table key, an index flag, a mirror flag, and a logical table name. The primary key is the actual table key. The actual table name corresponds to the physical name of this table in the database implementing the datamart 150. The actual table type is the logical type of this physical table. For example, if this is a dimension staging table or a fact staging table. The dimension base key points to the dimension base table definition that defined the corresponding physical table. The external table

key points to the external table definition that defined the physical table. The fact table key points to the fact table definition that defined the corresponding physical table. The index flag and the mirror flag are used in indexing and mirroring, respectively. The logical table name defines the logical name for this table.

5 The actual column 504 is metadata describing a physical column in a physical table in the datamart 150. The actual column table latches this definition information when the physical tables are built in the datamart 150. The actual column 504 includes the following attributes: the actual column key, an actual column name, an actual column type, an actual table key, a dimension role name, a foreign table key, a group by field, a hierarchy, a list order, a parent hierarchy, a physical type, a primary key, and a time navigation field. The actual column name is the name of the physical column in the physical table in the datamart 150. The actual column type is the logical type of the column. The actual table key points to the actual table in which the actual column lives. The dimension role name is the logical role name of the dimension in the fact table for dimension foreign keys inside of a fact table. The foreign table key points to the actual dimension base tables in the actual tables 502 (the foreign table key is applicable to fact actual columns that are foreign keys to dimensions). The group by field, for dimension table, is true when this column should be included in an aggregate builder group. The hierarchy for dimension, for dimension columns, indicates that aggregate builder group to which this column belongs. The list order is the order of the column in the actual table 502. The parent hierarchy, 15 for dimension columns, indicates the parent aggregate builder group to which this column belongs. The physical type is a logical data type of the column. The primary key, for dimension 20

tables, is true when this column is the primary key of the actual table 502. The time navigation field, for the database dimension, is true if this field can be used by the time navigator.

The fact aggregate table 512 includes a list of fact aggregates in the datamart 150. The fact aggregates includes attributes that point to the actual fact table in which this aggregate belongs.

- 5 The fact aggregate table 512 indicates which numbered aggregate represents the fact table in question, the number of rows in this aggregate, a datamart letter, and an enabled flag. The datamart letter indicates the mirrored datamart to which this fact aggregate belongs.

Mirror is used to ensure that partially completed extractions from the source systems 110 do not cause the database to become inconsistent.

10 The fact aggregate dimension 514 lists which aggregates contain which dimensions. The fact aggregate dimension includes the following attributes: an actual dimension role key, a dimension base aggregate key, a fact aggregate dimension key, and the fact aggregate key. The actual dimension role key is the dimension foreign key in this fact aggregate that is being defined. The dimension base aggregate key is the dimension aggregate that this fact aggregate points to for this foreign key. The fact aggregate dimension key is the primary key. The fact aggregate key points to the fact aggregate being defined.

The dimension base aggregate table 516 lists all the dimension aggregates in the datamart.

The dimension base aggregate includes the following attributes: an actual table key, an aggregate number, an aggregate size, a datamart letter, a dimension base aggregate key, and an enable flag.

- 20 The actual table key points to the physical header for this dimension base. The aggregate number, for the dimension table in question, is the number of this particular aggregate. The

aggregate size is the number of rows in the aggregate. The datamart letter indicates which mirrored database this aggregate lives in. The dimension base aggregate key is the primary key. The enable flag indicates whether or not the aggregate navigator should work with this aggregate.

The dimension base aggregate column 518 is a list of columns in a given dimension
5 aggregate. The dimension base aggregate column includes attributes which point to which column is included in this dimension aggregate, and a pointer to a dimension aggregate being defined.

The datamart letter 510 indicates which of two mirrored datamarts a particular aggregate belongs to. This is an optional element which may not be required if mirroring does not occur in the datamart 150. Mirroring duplicates the tables in the datamart 150. Changes can then be made to one copy of the datamart 150, while the other datamart 150 continues running. These changes can then be propagated when possible.

The actual column type 540 is a logical description of role a column plays in the system 100. The actual column type 540 includes attributes that define the default value to be used in a
15 database for a column of this type.

The physical type definition 530 defines which physical types are allowed for which table types. The physical type definition 530 includes attributes which point to an actual table type. The actual table type is a logical type of a physical table (for example, dimension, fact etc.) being defined. The physical type definition also includes an attribute that indicates whether to select
20 this item by default when giving the consultant or user a choice.

The database physical type 595 defines the name of the physical database.

The translation actual table 539 defines the actual values of translations strings for a single relational database management system. These translations strings are the real strings to use for a given translation string within a store type. The translation actual table 539 also includes attributes that point to the store type.

Figure 5 also illustrates additional tables used in the system 100.

The date table 560 is used to track date information in the datamart 150. Importantly, times and dates are always treated corrected in the datamart 150. This can be guaranteed because the consultant cannot change the definition of dates in the datamart 150. Thus, for example, the month of September will always have 30 days, and leap years will be handled correctly.

The transaction type table 590 is a list of the available transaction types within the system 100.

The adaptive template profile 580 is used as a communications mechanism for templates. The adaptive template profile 580 includes a number of rows being communicated back to the calling program. The adaptive template profile 580 also indicates the logical name for information being communicated back from an adaptive template to the calling program.

The business process table 570 is a look up table for supported business process types during the extraction. The business process 570 includes a business process key and a process name. The process name corresponds to a logical name for a business process to which fact staging table belongs. The process key identifies a business process record in a fact staging table.

Time Navigation

An important feature of some embodiments of the invention is the ability to compactly store and rapidly query "historical" backlog/balance/inventory quantities. By historical we mean the amount of backlog or inventory as it existed at a given point in time--not necessarily today. Note
5 that backlog/balance/inventory quantities are different than transactional quantities. For example, your bank balance at the end of Q1 1997 is not the sum of your bank balances at the end of Jan, Feb and March. It is computed by adding all of the deposit transactions and subtracting all the withdrawals from the balance at the end of Q4 1996. One could compute balances by this method when a user queries the system but because this method requires rolling
10 forward all of the appropriate transactions "from the beginning of time," the queries will likely be slow.

The traditional solution in datamarts is to store periodic "snapshots" of the balance. The snapshots are often stored at daily intervals for the recent historical past, and at greater intervals (e.g. weekly or monthly) for less recent history. This approach has two big disadvantages. The
15 first is an enormous multiplication of data volume. If, for example, you are keeping track of inventory in a store you must store a snapshot for each product you hold in inventory for each day, even if you only sell a small fraction of all of your products on a given day. If you sell 10,000 different products but you only have 500 transactions a day, the "snapshot" datamart is 20 times larger than the transactional datamart. The second disadvantage relates to the most
20 common solution for alleviating the first problem, namely storing snapshots at less frequent intervals for less recent history. This results in the inability to compare levels of inventory in

corresponding time periods since the same level of detail is not present in earlier data. For example, in manufacturing companies it is often the case that much business is done near the end of fiscal quarters. If one wants to compare inventory levels between Q1 1995, Q1 1996 and Q1 1997, and focus on the most important changes which occur near quarter end, one cannot use the approach of storing the snapshots at coarser levels of detail since daily data would be required.

In some embodiments of the system, the aggregate tables are used to answer queries about backlog/balance/inventory quantities. In order to answer such queries, the previously described rolling forward from the beginning of time is done. However, this is performed efficiently through the accessing of the appropriate time aggregates. For example, assume the datamart has five years of historical transaction data beginning in 1993. Assume that one desires the inventory of some specified products on May 10, 1996. This would be computed by querying all of the transactions in the 1993, 1994 and 1995 year aggregates, the 1996 Q1 quarter aggregate, the April 1996 month aggregate, the May 1996 week 1 aggregate and finally 3 days of actual May, 1996 daily transactions. These transactions (additions and subtractions from inventory) would be added to the known starting inventory in order to produce the inventory on May 10. Note that this "time navigation "hops" by successively smaller time intervals (year, quarter, month, week, day) in order to minimize the number of database accesses. What is important is the exploitation of aggregate tables, that already exist in the system in order to answer transactional queries rapidly (e.g. What were the total sales of product X in Apr 1996?). This avoids the need to build what is essentially a second data datamart with the balance/inventory/backlog snapshots.

Query Mechanism Metadata

The following describes the metadata 160 used in the query/reporting program 104. This metadata is shown in Fig. 6. Generally, the query mechanism metadata can be broken into ticksheet metadata, measurement metadata, filtering metadata and display options metadata. The ticksheet metadata defines the user interface objects for user interaction with the datamart 150. The ticksheet defines how users can initiate queries and how results are presented back to the user. The measurement metadata defines a logical business calculation that can be presented to a user. Typically, the measurement metadata defines a format for presenting information to user that is more easily understood by the user or provides a more valuable result to the user. The filtering metadata defines how a user can filter results. Filtering allows the results set to be limited to particular dimension values. The display options metadata defines display options that can be provided to the user.

The following describes some important features of the user interface. The user interface allows the user to drill down through data. Also, portions of the query forms can be dynamic based upon values in fields (e.g., a list box can be dynamically updated because it is tied to a field in the datamart 150, that when changed, cause the values in the list box to change). Also, a query is guaranteed to be consistent with the schema because the query is tied to the schema definition.

Query Mechanism Schema List.

Figure 6 includes the following elements: the constellation 302, a ticksheet 602, a data set 606, a ticksheet column 608, a tip 601, an attribute role 603, an attribute 610, a ticksheet attribute 605, a ticksheet type 604, a measure 620, a measure term 630, a measure unit 624, a term operator 632, a transaction type 590, an RPN operator 636, the fact column 310, the fact table 304, a backlog 638, a measure mapping 622, a ticksheet column element 612, a dictionary 640, a filter block 650, a filter block type 652, a filter group 654, a filter element 656, a ticksheet type option 660, an option location 662, an option value 664, an option name 666, an option display type 668, an application type 691, the dimension role 320, the dimension column 329, and the dimension base 306.

Query Mechanism Schema Metadata Descriptions

Ticksheets Metadata

Under the constellation 302, the ticksheet is a top level object for defining the user interface objects for user interaction. The ticksheet 602 table includes a data set key, a name, a ticksheet type, a constellation key, a label, label detail, a list order, a cleanse flag, and a description. The cleanse flag indicates whether or not to cleanse the filter data on this ticksheet. The constellation flag indicates the constellation in which the dimensions and measures for this ticksheet reside. The data set key indicates the page in which the end user makes report selections. The data set key represents a logical grouping of similar ticksheets. The description is for documentation purposes. The label is the string for the name of the ticksheet. The list order indicates the ticksheet list order. The name is the hidden name of the ticksheet. The ticksheet includes a

primary key. The label detail allows for more verbose documentation. The ticksheet type indicates the type of application that interprets the selections made on this ticksheet.

The data set table 606 is a grouping mechanism for ticksheets into sets that describe their contents. The data set table 606 includes the following columns: a data set key, a data set name, a label, a description, and a list order. The data set name is a logical name for top level user definition of like ticksheets across ticksheet types. The description is the description of the data set.

The ticksheet column 608 defines a single column for displaying measure choices on a ticksheet. The ticksheet column table 608 includes the ticksheet column key, the list order, the ticksheet key, and the description. These columns and the ticksheet column table 608 operate in a manner similar to such columns in other tables in this metadata.

The ticksheet column element 612 is a single value within a measure display column on a ticksheet. The ticksheet column element 612 includes an abbreviation, a description, a dictionary key, a label, a list order, a name, a ticksheet column element key, and a ticksheet column key. The ticksheet column element key is the primary key for this table. The ticksheet column key points to the ticksheet column 608 entry. The name is the hidden name for the column element. The abbreviation is the shortened name for the user display. The dictionary key is the key for help text entry for this element. It is a key into the dictionary 640. The other columns act in a manner similar to columns in other tables of similar names.

The tip table 601 includes a ticksheet key, a name, a description, and a list order. The tip is a definition of user tips for using a particular ticksheet.

The attribute table 610 defines the dimension attribute choices within a ticksheet. These choices are tied to a single dimension column in the schema definition of the datamart 150. The attribute table 610 includes an abbreviation, an attribute key, a dictionary key, a dimension column key, a dimension role key, a hyperlink, a label, a list order, a name, and a ticksheet key.

5 The abbreviation is the shortened user string for the attribute. The attribute key is the primary key for this attribute. The dictionary key is a pointer to the dictionary 640 that includes help message for a particular attribute. The dimension column key is the dimension column in which this attribute refers. For degenerate dimensions this reference is null. The dimension usage, within a constellation, is defined by the dimension role key. The hyperlink is an html text for navigating return values for this attribute to other web sites, such as a company name look-ups etc. The label is what the user sees for a particular attribute. The list order defines a sort order on pop-up menus where one is the topmost in the list. The name is the internal name for the attribute. The ticksheet key indicates the ticksheet to which this attribute belongs.

10 The attribute role 603 table is a lookup table list of valid roles for attributes within a ticksheet type. The attribute role includes the attribute role key and the ticksheet type.

15 The ticksheet attribute 605 indicates the roles played by dimension attributes within a ticksheet. The ticksheet attribute 605 includes the attribute key, an attribute role, a ticksheet attribute key, and a ticksheet key. The attribute key indicates the attribute in the attribute table 610 which has a role define on the ticksheet. The attribute role is the role being granted. The ticksheet attribute key is the primary key. The ticksheet key is the ticksheet being defined.

20 The application type 691 is the top level grouping for ticksheet types.

The ticksheet type table 604 lists the applicable applications that can use ticksheets.

Examples may be simple reporting applications or relevancy applications or trend type of applications. The ticksheet type 604 includes the application type, a ticksheet type, a template, and a ticksheet type name. The application type is the definition of the high level application in which a particular ticksheet type resides. The ticksheet type is the logical name for a program that interprets ticksheets. The template is the template for the ticksheet. The ticksheet type name is the name displayed for a particular ticksheet type.

Measurement Metadata

The following describes the measures used in the query mechanism schema 600. The measure table 620 defines a top level object for a logical business calculation. The measure table 620 includes a constellation key, a description, a measure key, a measure unit, and a name. The constellation key points to the constellation in which the measure resides. The description is for documentation purposes. The measure key is the primary key for the measure table 620. The measure unit is an indicator of the manner in which numbers are to be displayed. The name is the logical name of the measure.

The measure unit table 624 is a lookup table of the valid list of measure unit types. An example of such unit types is currency.

The measure term table 630 indicates a single component of a measure. The measure term can be combined arithmetically to construct a measure. The measure term table 630 includes a backlog type, a fact column key, a fact table key, a list order, a measure key, a measure term key, an RPN operator, a term operator, and a transaction type key. The backlog type indicates the

type of backlog operation to use for a particular term (e.g., "beginning of period" and "end of period"). This can possibly be none. The backlog types are defined in the backlog type table 638. The fact column key points to the particular numeric column to operate on in the fact table. The fact table key indicates the fact table being operated on. The list order is the order of this term in the measure 620. The measure key is the measure being defined. The measure term key is the primary key for this table. The RPN operator is for the measure terms that perform arithmetic operations on other terms. (The RPN operator table lists the valid arithmetic operations to use when constructing a measure.) The term operator is an SQL operator to use on a set of fact rows. (The term operator table 632 indicates the valid set of SQL operators to use on a measure term.) The transaction type is the transaction type values to filter on for the fact in question.

The relation between measures and ticksheets is handled through the measure mapping table 622. The measure mapping table 622 includes the measure key, the ticksheet key, and the combination ID. The measure key points to the particular measure that is related to the particular ticksheet. The combination ID identifies a set of ticksheet column elements being defined.

Filtering Metadata

Filtering allows results to be limited to only particular dimension values. For example, a user may want to limit the results to particular customer names.

The following describes the filtering tables. The filter block 650 is a top level grouping table for filter area within a ticksheet. The filter block 650 is tied to a particular dimension column in the schema definition. The filter block 650 includes, columns, a description, a dictionary key, a

dimension column key, a dimension role key, a filter block key, a filter block type, a label, a list order, a name, a plural, a mapping flag, and a ticksheet key. The columns field indicates the number of columns in this filter block. The description is for documentation. The dictionary key points to the help dictionary. The dimension column key points to the actual column name and the datamart to be filtered on. A null value here means degenerate dimension as determined by the dimension role key. The dimension role key points to the dimension role in the constellation of the ticksheet that is the form key to filter on for all facts in this constellation. A null value here means that a special dimension shared by all constellations is being used. The filter block key is the primary key for this table. The filter block type points to the filter block type table 652 which defines the ways in which this filter block is displayed to the user (e.g., a checkbox or a radio button). The label is the text that appears to the user for the filter block. The list order is the order that the filter block should appear in a list. The name is the name of the filter block. The plural field is the text that appears to the user for the filter block. The mapping flag is used in mapping. The ticksheet key points to the ticksheet that this filter block belongs.

The filter group 654 is a mid-level grouping for a filter on a ticksheet. This groups individual selections into logical units.

The filter element table 656 defines individual values for a dimension attribute within a filter block. The filter element table 656 includes a dictionary key, a filter element key, a filter group key, a label, a list order, a name, an SQL statement, and a value. The dictionary key points to the user help text for a particular filter element. The filter element key is the primary key. The filter group key points to the filter group to which this element belongs. The label is the user

displayed string for the element. A list order is the order of this element within a filter group.

The name is the hidden name of this element. The SQL statement is an SQL statement used to build the list of values for a dynamic list box filter group. The value is the database value that this element translates into in a SQL "WHERE" clause.

Display Options Metadata

To control the method in which information is displayed with a ticksheet, a set of options are supplied. The ticksheet type option table 660 helps support this feature. The ticksheet type option table 660 includes a list order, an option display type, an option location, an option name key, a ticksheet type, and a ticksheet type option key. The list order is the order to display options for a ticksheet type. The option display type is an html control type to use when displaying a particular option. The option location is the location of the option on the ticksheet.

(The option location table 662 holds the list of possible locations of options on a ticksheet.) The option name key is the option being included in a ticksheet type. (The option name table 666 defines an option that has meaning for one or more applications that can be used by users.) The ticksheet type is the ticksheet type being defined. The ticksheet type option key is the primary key for this table.

The option name table 666 includes option name key, a name, a label, and a dictionary key. The option name key is the primary key. The name is the logical name for the option. The label is the label seen by the user as the name of the option. The dictionary key is the pointer to the help text dictionary.

The option value table 664 defines single valid values for options. The option value table 664 includes the dictionary key, a label, a list order, an option name key, an option value key, and a value. The dictionary key is the help text dictionary key. The label is the label seen by the user for this option choice. The list order is the order of the valid values for the option. The option name key is the option set being defined. The option value key is the primary key for this table. The value is the hidden value for the option.

The option display type table 668 is a lookup table indicating the valid way that options can be displayed.

The dictionary table 640 is a table for help text for users.

User Interface Example of Defining Metadata

General Schema Definitions User Interface

The following describes a constellation used in a business. In this example a new dimension is added very simply and the changes are automatically propagated into the datamart 150. The enterprise manager interface 192 is used by the consultant to define and manipulate the system 100.

Figure 7 illustrates the enterprise manager interface 192. Multiple system 100's can be connected to through that interface. Many of the objects and tables in the system 100 are shown. The base dimensions definitions 710 correspond to the base dimensions available under the "epitest" datamart. The constellations 712 for this datamart include an expense constellation and a sales constellation 720. Thus, the sales constellation 720 would appear as a row in the

constellation table 302. Under the sales constellation 720 appear the definitions for the sales aggregates 721, the sales dimensions 723, the sales degenerate dimensions 725, the sales facts 726, the sales measures 728, and the sales ticksheets 729. Also, the extraction definitions 740 and security definitions for the "epitest" datamart are accessible. The sales dimensions 723
5 define rows in the dimension role table 320. These rows include customer billed to, product, application, program, customer ship to, and territory.

Figure 8 illustrates the dimension table definition window 800 (presented to the consultant as the result of selecting the customer billed to dimension role under dimensions). A dimension table definition window 800 show that the dimension 820 is customer bill to and the base dimension 810, to which the dimension 820 points to, is named customer.

Figure 9 illustrates a base dimension window 900 showing the definition of the base dimension 810 named customer. In this case, the customer base dimension has a "slowly changing dimensions" dimension data semantic 910. In this example, the dimension base 810 customer has a number of dimension columns 920. L1 is an example of a dimension node.

Figure 10 illustrates the dimension column window 1000 for the customer region code column 1010. The physical type is the type of data defining that dimension column. The VARCHAR_50 physical type is then mapped to an actual type through the physical type table 330. The translation is dependent on the store type.

Figure 11 illustrates the base dimension window for the base dimension date (substantially un-editable). The user interface indicates that the date dimension is not an editable base dimension (shown as black circles under "Base Dimensions"), and grayed out in the base

dimension window 900. The transtype (transaction type) is similarly not editable and is similarly shown in the user interface.

Figure 12 shows the dimension column "date day quarter end". Note that column cannot be edited.

5 Figure 13 illustrates a fact table window 1300 that is open on the order fact table 1310 definition. The fact data semantic 1310 is transactional/state like/force close/unjoined. The transactional/state like/force close/unjoined means that the invoice part of an order is transactional, the booking is state like, orders that are not otherwise dealt with, are closed out, and the data may become dirty and so it needs to be cleansed, thus, it is unjoined. This semantic
10 type is described in detail in Appendix A. Note that the user can select from many different types of fact table semantics. The fact table window 1300 also shows the fact columns 1330 for the order fact table.

Figure 14 illustrates a fact column window 1400 opened on the definition of the net_price
15 fact column 1410. Here the fact column 1410 has a physical type 1420 called FACTMONEY and an aggregate operation 1430 called a SUM.

Figure 15 illustrates how the consultant would select the semantic type for the order fact table 1310.

Figure 16 illustrates the results of a request by the consultant to generate the datamart 150
from the definitions of the datamart. The results show that a number of tables have been created
20 in the datamart 150. Importantly, Figure 16 illustrates the results of an initial build process. In subsequent modifications, only those elements of the datamart that have changed will be

changed. In other words, the subsequent changes are handled as an update process. An example of the update process is described below.

Extraction Interface Elements

The following describes the creation of the connectors 162. Once the schema definitions 161 are set, the consultant then defines the connectors 162 to the source systems 110. The connectors, as noted above, define how information is to be extracted from the source systems 110 and how that information is to be placed into the datamart 150.

These connectors are defined under the extraction definitions 740. Figure 17 illustrates the job definition window 1700 presented when the consultant has selected a particular job.

Figure 18 illustrates the job steps 1810 within the default job. The checkbox indicates whether the particular job step is enabled for that job. The list of job steps is shown in the order that they are executed. The two foreign keys within the job step are shown in the dialog box of Figure 17 to indicate whether the job step is a connector or a system call.

Figure 19 illustrates the All Semantics connector as defined in the connector definition window 1900. This connector includes the description and a definition of the input and output data stores. In this case, both of the data stores are the "epimart" (which is the datamart 150).

Figure 20 illustrates the data store window 2000 interface for showing a data store. This is the data store that is referenced in the connector All Semantics.

Returning to the discussion of connectors, Figure 21 illustrates the connector entitled MFG. The MFG connector has two major steps: (1) order dimension staging, and (2) order fact staging.

The results of these extraction steps are put in the staging tables 130. (The all extraction steps window 2100 illustrates all the possible steps in the system that can be used.)

Figure 22 illustrates the SQL statement window 2200. The SQL statement window 2200 has an SQL field 2210 that includes the SQL statements that loads a customer table. As shown in the dialog box, the table references for the SQL statement includes the customer dimension table column definitions. That is, this SQL statement is going to be used to populate the customer dimension table.

In this example, the base name, type code, type name, region code, region name and tier name corresponds to the column names within the customer dimension. The date modify is an additional field that is to be used to indicate when this field was last modified in the database. Additionally, there is a source system key that is automatically included in every dimension. The source system key helps ensure that the datamart 150 is well-formed.

In one embodiment of the invention, these names can be automatically propagated into the SQL field 2110 window via a template that is generated from the corresponding base dimension. This allows the consultant to more easily define the SQL selection statement.

Figure 23 illustrates the SQL statement for the Open Order Stage for populating the order fact table.

At this point the steps for generating the staging table information are complete. Now the semantic conversion steps are defined.

In Figure 24, returning to the connector steps window, we have switched to an All Semantics connector 2410. The All Semantics connector 2410 causes the semantic conversion of the information in the staging table for use in the datamart 150.

Figure 25 illustrates the semantic transformation window 2500 showing the dimension table customer semantic 2510.

Figure 26 illustrates the order fact semantic 2610 definition.

Figure 27 illustrates the results of a consultant adding a new dimension 2700 (called warehouse) to the sales constellation 720. The batch operation window 1600 illustrates the changes that are being made to the datamart that was created in Figure 16. To achieve these results, the consultant need only perform the following steps:

1. Define the new dimension.
2. Define the connector steps, including the SQL Statement to extract the warehouse data from the source systems 110.
3. Add the warehouse information to the Open Order Stage SQL Statement.
4. Define a semantic transformation for the warehouse, e.g., slowing changing dimension.
5. Have the enterprise manager 102 update the datamart 150.

Thus, changing the schema definition of the datamart 150 is significantly simpler than previous systems.

Additional Interface

Figure 28 illustrates the aggregate group window 2800, where aggregates can be defined.

For a given aggregate group, the consultant can define which fact share the aggregate, and which

type of aggregate should be built for a given dimension in the aggregate. Additionally, dimensions can be added to, or removed from, an aggregate group.

Figure 29 illustrates a portion of the configuration window 2900. In this example, a partial list of the transaction types 2910 is shown. Thus, the consultant can determine which transaction types will be available to him/her.

End User Interface Definition and Example

Figure 27 illustrates the interface used to define a user interface for the end user. Figure 30 includes a user interface definition window 3000 which can be used to define measures and ticksheets. In this case, the measure definition window 3010 is shown.

The measure definition window 3010 allows the consultant to define which measures will be available in the system. The consultant defines the name, units, and constellation for a particular measure. The measure is further defined by defining the list of measure terms that make up a measure (the calculations for the measure 3020). In this example, the ASPBacklogOrderGross measure has seven calculation steps, some of them arithmetic (e.g., SUM) and others RPN (Reverse Polish Notation).

Figure 31 illustrates the ticksheet definition window 3100. The ticksheet definition window allows a consultant to define a ticksheet that will be used to generate a query form for a user. The consultant defines the attributes, the columns, and the filters for a ticksheet. Figure 32 illustrates the query form 3200 generated from the ticksheet defined in Figure 31.

Figure 33 illustrates the measure mappings window 3300, that allows the consultant to map measure definitions to user friendly measure names. In the example of Figure 33, the

PriceShipGrossMonth measure is mapped to a combination of the dollar amount, gross, and sell-through being selected in the query form 3200.

Figure 34 illustrates another query form 3200 generated from a different ticksheet definition.

When the user selects the create report button, the query is issued against the datamart 150.

5 Figure 35 illustrates some sample results from such a query.

The following query log illustrates the actual query that was executed against the datamart 150. The query log illustrates that an aggregate and navigation process determined which aggregate would be the most appropriate. The aggregate builder had created these aggregates. The most appropriate aggregate for the requested query was selected. The results were then returned.

```
*****
Query log
*****

time      : <A Date Here>
addr      : 192.0.0.210
host      : 192.0.0.210
user      :
agent     : Mozilla/4.01 [en] (WinNT; U)

Database information: DRIVER={SQL SERVER};SERVER=bigfoot;DATABASE=macromedia

Keys and values coming in from the browser:
  file =
  fileDesc =
  queryaction = QUERY
  hidden_queryaction = QUERY
  OK_callback =
  NOK_callback =
  ticksheet = Orders
  hidden_ticksheet = Orders
  Rows = Customer
  hidden_Rows = Customer
  Columns = Fiscal Year
  hidden_Columns = Fiscal Year
  units = Price
  hidden_units = Price
  facttype = Shipped
  hidden_facttype = Shipped
  facttype2 = Gross
  hidden_facttype2 = Gross
  stage = Orders
  hidden_stage = Orders
  currencyunits = Thousands
```

```

hidden_currencyunits = Thousands
rowtotal = yes
hidden_rowtotal = yes
columnntotal = yes
hidden_columnntotal = yes
percent = none
hidden_percent = none
precision = 0
hidden_precision = 0
charts = 3D
hidden_charts = 3D
maxrows = 10
hidden_maxrows = 10
rowsorttype = value
hidden_rowsorttype = value
Fiscal_Years = All
hidden_Fiscal_Years = All
Fiscal_Quarters = All
hidden_Fiscal_Quarters = All
Calendar_Months = All
hidden_Calendar_Months = All
Business_Units = All
hidden_Business_Units = All
Product_Lines = All
hidden_Product_Lines = All
Product_Supergroups = All
hidden_Product_Supergroups = All
Platforms = All
hidden_Platforms = All
Product_Languages = All
hidden_Product_Languages = All
Product_SKUs = All
hidden_Product_SKUs = All
Product_SKU =
Sales_Reps = All
hidden_Sales_Reps = All
Channels = All
hidden_Channels = All
Customer_Types = All
hidden_Customer_Types = All
Customer_Regions = All
hidden_Customer_Regions = All
Customers = All
hidden_Customers = All
Customer =
sqStyle = classic
hidden_sqStyle = classic

```

Contents of %FormData:

```

Columns = Fiscal Year
rowsorttype = value
ticksheet = Orders
Customers = All
charts = 3D
Customer Types = All
Product SKUs = All
Customer Regions = All
Fiscal Quarters = All
Rows = Customer
Channels = All
precision = 0
Product Supergroups = All
percent = none
maxrows = 10
Sales Reps = All
columnntotal = yes
Calendar Months = All
queryaction = QUERY
sqStyle = classic
Fiscal Years = All
Product Languages = All

```

Platforms = All
Product Lines = All
rowtotal = yes
currencyunits = Thousands
Business Units = All

The colheaders are:

The Cellitems are:

Price Shipped Gross Orders

The Cellitems abbreviated are:

Dollar Amount Shipped Gross Orders

pid is: 310

spid is: 25

The valid collheaders are:

The invalid colheaders are:

The valid cellitems are:

Price Shipped Gross Orders

The invalid cellitems are:

The unitstack is:

CURRENCY

The cellstack is:

-SUM(Order.net_price)

The selectstack is:

-SUM(Order.net_price)

The typestack is:

SHIP

Transtypes are:

"BEGIN RETURN" = 1007
"END GROSS" = 1004
"END_SRBOOTH" = 1018
"END_SRADJ" = 1012
"BOOK" = 1
"BEGIN_ANET" = 1013
"END_IADJ" = 1024
"END_ICOMP" = 1022
"BEGIN_GROSS" = 1003
"BEGIN_SRBOOTH" = 1017
"END_SBOOTH" = 1016
"END" = 1002
"BEGIN_SRADJ" = 1011
"END_SADJ" = 1010
"BEGIN_IADJ" = 1023
"BEGIN_ICOMP" = 1021
"END_NET" = 1006
"SHIP_ADJUST" = 103
"LOST" = 3
"END_SALL" = 1020
"BEGIN_SBOOTH" = 1015
"BEGIN" = 1001
"BEGIN_SADJ" = 1009
"BOOK_RETURN" = 2
"END_FADJ" = 1026
"BEGIN_NET" = 1005
"BEGIN_SALL" = 1019
"GL" = 105
"SHIP_RETURN" = 102
"SHIP_RADJ" = 104
"SHIP" = 101
"END_RETURN" = 1008
"INV_ADJUST" = 201
"LEAD_LOST" = 4
"BEGIN_FADJ" = 1025
"FINV_ADJUST" = 202

"END_ANET" = 1014
The facttable is:
Order
The limitvalues are:
The access limitations are:
The limitvalues are:
The header took 261 milliseconds.
Parsing took 160 milliseconds.
Generating the header took 0 milliseconds.
Building user limits took 0 milliseconds.

Phase 0: Aggregate navigator initialization.
Phase 1: Getting dimensions from SQL. (10)
Phase 2: Getting fields available from SQL. (90)
Phase 3: Getting degenerate fields from SQL. (0)

Phase 0: Preparing for query building and execution.R2[0 -> 0] = (SUM(Z0))
R1[0] = SUM(Z0)

The column router:
Cell location 0 will be returned in column 0 when Type is SHIP.
The result router:
Result location 0 is (SUM(Z0)) 0

The unique facttables are:Order

The number of unique facttables are: 1

The unique types are:SHIP
Table to unique number lookup
Order => _0_

Begin work on the query based on the facttable Order
Phase 1: Table Order. Creating table aliases (10)
JOIN_FIELD IS CUSTOMER_BILLTO_KEY FOR Customer
JOIN_FIELD IS FOR Fiscal Year

SQL table aliases:
Order□ : T3
Date□ : T2
Customer□CUSTOMER_BILLTO_KEY : T1

Table aliases:
tablealiaslookup(T1) = Customer
tablealiaslookup(T2) = Date
tablealiaslookup(T3) = Order

selectalias:
Customer: T1.base_name
Fiscal Year: T2.fy_name

selectstackalias:
-SUM(Order.net_price): -SUM(T3.net_price)

joinalias:
Customer: T1.customer_key = T3.customer_billto_key

Phase 2: Building SELECT clause (0)
Phase 3: Building FROM clause (0)
Phase 4: Building WHERE clause (0)
Phase 5: Building GROUP BY clause (0)

SQL before going through the aggregate navigator:
SELECT
Columns = T2.fy_name,
Type = T3.Transtype_key,
C0 = -SUM(T3.net_price),
Rows = T1.base_name
INTO #tmp_0_
FROM
Customer T1,
Date T2,
Order T3

WHERE

T1.customer_key = T3.customer_billto_key and
T2.date_key = T3.date_key and
T3.Transtype_key in (101)

GROUP BY

T1.base_name,
T2.fy_name,
T3.Transtype_key

Selecting appropriate aggregate for the query.

Phase 0: Aggregate navigator. Preparing for query building and execution.

Phase 1: Splitting query into clauses. (0)

Phase 2: Construction of aliases. (0)

Phase 3: Extracting neededfields from where clause. (0)

Phase 4: Extracting neededfields from group by clause. (0)

Phase 5: Extracting neededfields from select clause. (0)

Phase 6: Unaliasing. (0)

Phase 7: Constructing the SQL to fetch smallest aggregate. (20)

Phase 8: Running the big SQL. (20)

Phase 9: Extracting results from the big SQL. (0)

Phase 10: Adjusting input with aggregate information. (0)

Phase 6: Aggregate Navigating (40)

Appropriate aggregate determined (CUSTOMER_0, DATE_4, ORDER_86), now select

SQL after going through the aggregate navigator:

SELECT

Columns = T2.fy_name,
Type = T3.Transtype_key,
C0 = -SUM(T3.net_price),
Rows = T1.base_name

INTO #tmp_0_
FROM

CUSTOMER_0 T1,
DATE_4 T2,
Order_86 T3

WHERE

T1.customer_key = T3.customer_billto_key and
T2.date_key = T3.date_key and
T3.Transtype_key in (101)

GROUP BY

T1.base_name,
T2.fy_name,
T3.Transtype_key

Phase 7: Building results table in sql (10435)

Phase 15: Splitting tables by type (needed = 0) (0)

Phase 16: Merging results into one table (needed = 0) (0) GR_COLS = C0 = SUM(C0)

SQL: SELECT Rows INTO #tmpAllRows FROM #tmp_0_ GROUP BY Rows ORDER BY SUM(C0) DESC

SQL: SELECT count(Rows) FROM #tmpAllRows

Phase 17: Extracting rows and doing number of total records (10508/10498) (1022)

SQL: set rowcount 10

SELECT Rows INTO #tmpTopRows FROM #tmpAllRows

set rowcount 0

Phase 19: Sorting, Top (needed = 10498) (10)

-- creating row totals

SELECT

#tmp_0_.Rows,
C0 = -SUM(C0)

INTO #tmpRows

FROM #tmp_0_, #tmpTopRows

WHERE #tmp_0_.Rows = #tmpTopRows.Rows

GROUP BY #tmp_0_.Rows


```

-- creating col, grand totals
SELECT
  Columns,
  C0 = SUM(C0)
INTO #tmpColumns
FROM #tmp_0
GROUP BY Columns

Checking ADJ of GRAND

SQL: SELECT C0 = SUM(C0) INTO #tmpGrand FROM #tmpColumns

-- final results table
SELECT #tmp_0.Rows, Columns, C0
INTO #tmpFinalResults
FROM #tmp_0, #tmpTopRows
WHERE #tmp_0.Rows = #tmpTopRows.Rows

Phase 20: Filtering results ( 841 )
Phase 21: Reading Row Totals ( 10 )
Phase 22: Reading Column Totals ( 10 )
Phase 23: Reading Grand ( 10 )
SQL:
-- calculate remaining columns
SELECT
  Columns = #tmpColumns.Columns,
  C0 = #tmpColumns.C0 - ISNULL(SUM(#tmpFinalResults.C0),0)
INTO #tmpRemaining
FROM #tmpFinalResults, #tmpColumns
WHERE #tmpColumns.Columns = #tmpFinalResults.Columns
GROUP BY #tmpColumns.Columns, #tmpColumns.C0
select C0 = SUM(C0) from #tmpRemaining

Phase 24: Reading Remaining Column Totals (needed = 10498) ( 20 )
Phase 25: Final Results ( 30 )
Phase 26: Sorting columns by time (if necessary). ( 20 )
Phase 27: sorting rows by time or name (if necessary). ( 0 )ERR FROM BUILD_AND_EXEC:0

Getting results took 12658 milliseconds.

Phase 0: Begining output generation.
Phase 3: performing cumulative (if necessary). ( 0 )

Dollar Amount / Shipped / Gross
The columns are: 1994
                  1995
                  1996
                  1997
                  1998

The rows are: ***** A number of customer names here *****

The table headers are:
The contents of the results array are:
*****
A number of customer name, value pairs
*****

The contents of the rowtotal array are:<
key: *****Row totals*****<
...

The contents of the coltotal array are:
1995: *****An amount*****
1996: *****An amount*****
1997: *****An amount*****
1998: *****An amount*****
1994: *****An amount*****

The contents of the grdtotal array are:
Grand: *****A grand total amount*****

Processing and formatting results took 220 milliseconds.

```

Total time was 13299 milliseconds.
Processing sylk took 81 milliseconds.

Figure 36 illustrates the options form 3600 that the user can use to select the display options for a result.

Alternative Embodiments

5 The following describes alternative embodiments of the invention.

Importantly, various embodiments of the invention do not necessarily include all of the features described above. For example, some embodiments of the invention do not include the first phase of the extraction process (loading the staging tables) because the source system data is provided to the system 100 directly by other extraction programs. Another example is where the datamart 150 is created, but a separate query interface is used to query the datamart 150. The query interface could use only a different communications protocol (e.g., instead of HTTP), or could be a completely different front end.

Other embodiments of the invention are configured differently than the embodiments described above. For example, the extraction node key in the semantic instance table 308 is not
15 included, but the extraction node 410 includes a semantic instance key.

Some embodiments of the invention build a database system, not necessarily a datamart. Additionally, these embodiments do not have to conform to a star schema definition in the metadata 160.

An object database system could be generated instead of a relational database system.

Some embodiments of the invention comprise only a computer readable media (e.g., a CD, a tape, a hard drive or other storage media) that has the programs that implement all, or a portion of, the system 100. Some embodiments of the invention include an electromagnetic waveform having the programs. Some embodiments of the invention include only the computer system running the datamart, other embodiments of the invention include only the computer system that creates, accesses, and queries the datamart, but does not include in the datamart itself.

Additional, or different, data semantics can be included in other embodiments of the invention.

Some embodiments of the invention include different user interfaces for the enterprise manager interface 192 and the query/results interface 184. For example, in the enterprise manager interface 192, the semantic types need not be selected in the fact and dimension base windows, but can be selected in the semantic instance window.

Attached hereto as Appendix B are 8 pages of Epiphany Software Release Notes entitled "Software Release 3.2: Clarity Update, New Product: Relevance" (Revision 1.16, August 7, 1998).

Attached hereto as Appendix C is "Epiphany System Guide, Clarity and Relevance 3.2", dated July 1998 (314 pages).

Attached hereto as Appendix D is "Epiphany Installation Guide 3.4", dated January 1999 (52 pages).

Attached hereto as Appendix E is "Epiphany System Guide 3.4, Service Pack 1", dated March 1999 (378 pages).

Attached hereto as Appendix G is "E.piphany E.4 System Guide, Release 4.0", dated June 1999 (544 pages).

Attached hereto as Appendix G is "E.piphany E.4 System Guide, Release 4.0", dated June 1999 (544 pages).

[illegible]

The Claims

What is claimed is:

1. A method of creating a system for creating a well-formed database system using a computer, the method comprising:

the computer accessing a definition of the system, the definition defining a schema for use by the system, the schema defining a set of tables, a set of columns that correspond to the set of tables, and a set of relationships between the tables of the set of tables, the definition further defining a set of operations for manipulating the data, the set of operations defining programs that operate on the set of tables and the set of table columns; and

the computer using the definition to generate the set of tables.

2. The method of claim 1 wherein the set of tables includes a first table and a second table, wherein the first table includes a first column, wherein the second table includes a second column, and wherein the first column and the second column are related by a join and are therefore guaranteed to be from the same domain.

3. The method of claim 1 wherein the set of tables includes a first table and a second table, and wherein the definition defines that the first table relates to the second table by a many to one relationship, and wherein the generating the set of tables includes automatically generating a

foreign key column in the first table, wherein the foreign key column is for holding a foreign key to the second table.

4. The method of claim 1 wherein the set of tables includes a first table and a second table, and wherein the definition defines that the first table relates to the second table by a many to many relationship, and wherein the generating the set of tables includes automatically generating an associative table corresponding to the first table and the second table, and wherein the associative table has a unique value created for each unique many-to-many relationship between the first table and the second table.

5. The method of claim 1 wherein the set of tables includes a first table and a second table, and wherein the first table includes one or more columns from the second table, and wherein said one or more columns are automatically populated from the one or more columns.

6. The method of claim 1 wherein the computer using the definition to generate the set of tables also includes the computer performing at least some of the set of operations on at least some of the set of tables.

7. The method of claim 1 wherein a transaction type column is automatically included in some tables of the set of tables.

8. The method of claim 1 wherein a date column is automatically included in some tables of the set of tables.

9. The method of claim 1 wherein a source system key column is automatically included in some tables of the set of tables.

10. The method of claim 1 wherein the definition defines a set of source system extraction operations, wherein the set of source system extraction operations are for extracting data from a source system and for manipulating the data for populating the database, and wherein the set of source system extraction operations correspond to the schema definition.

11. The method of claim 10 wherein the source system extraction operations correspond to the schema definition by populating source system data into the database system according to the schema definition.

12. The method of claim 1 wherein the definition defines a set of aggregates for the database system, the set of aggregates corresponding to the schema definition, the method further comprising:

the computer using the definition to create a set of aggregate tables corresponding to the set of aggregates; and
populating the set of aggregate tables.

13. The method of claim 12 wherein the set of aggregates corresponds to the schema definition by defining which aggregates should be made from which tables in the database system.

14. The method of claim 12 wherein the definition defines an aggregate operation for an
5 aggregate of the set of aggregates.

15. The method of claim 14 wherein the aggregate operation includes a SUM operation.

16. The method of claim 14 wherein the aggregate operation includes an AVERAGE operation.

17. The method of claim 1 wherein the definition includes a user interface definition for querying the database and for presenting results, the user interface definition corresponding to the schema definition.

18. The method of claim 17 wherein the user interface definition specifies which columns from which tables can be used in a query.

19. The method of claim 1 wherein the definition defines a set of source system extraction
15 operations, a set of aggregates, and a user interface definition, that correspond to the schema definition.

20. The method of claim 1 wherein the database system includes a datamart, wherein the schema definition includes a star schema definition, wherein the set of tables includes a set of fact tables and a set of dimension tables.

21. A system comprising:

5 a database system;

a first program for accessing a definition of the schema for the database system, the schema defining a set of tables, a set of columns corresponding to the set of tables, and a set of relationships between the tables of the set of tables, the definition further defining a set of operations for manipulating the data, the set of operations defining programs that operate on the set of tables and the set of table columns, the first program further for using the definition to generate the set of tables.

22. The system of claim 21 wherein the set of tables includes a first table and a second table, wherein the first table includes a first column, wherein the second table includes a second column, and wherein the first column and the second column are related by a join and are therefore guaranteed to be from the same domain.

23. The system of claim 21 wherein the set of tables includes a first table and a second table, and wherein the definition defines that the first table relates to the second table by a many to one relationship, and wherein the generating the set of tables includes automatically generating a

foreign key column in the first table, wherein the foreign key column is for holding a foreign key to the second table.

24. The system of claim 21 wherein the set of tables includes a first table and a second table, and wherein the definition defines that the first table relates to the second table by a many to many relationship, and wherein the generating the set of tables includes automatically generating an associative table corresponding to the first table and the second table, and wherein the associative table has a unique value created for each unique many-to-many relationship between the first table and the second table.

25. The system of claim 21 wherein the set of tables includes a first table and a second table, and wherein the first table includes one or more columns from the second table, and wherein said one or more columns are automatically populated from the one or more columns.

26. The system of claim 21 wherein the first program includes an enterprise manager for accessing the definition, causing the generation of the set of tables, and causing the population of the tables.

27. The system of claim 21 further comprising a database, the database for storing the set of tables.

28. The system of claim 21 further comprising an aggregate building program for accessing a definition of a set of aggregates and the definition of the schema and for generating the set of aggregates from the definition of the set of aggregates and the definition of the schema.

29. The system of claim 21 further comprising a query and reporting program for generating
5 a user interface from a definition of the user interface and the definition of the schema.

30. A system comprising:

means for accessing a definition of the system, the definition defining a schema for use by
the system, the schema defining a set of tables, a set of columns corresponding to the set
of tables, and a set of relationships between the tables of the set of tables, the definition
further defining a set of operations for manipulating the data, the set of operations
defining programs that operate on the set of tables and the set of table columns; and
means for using the definition to generate the set of tables.

31. The system of claim 30 wherein the set of tables includes a first table and a second table,
wherein the first table includes a first column, wherein the second table includes a second
15 column, and wherein the first column and the second column are related by a join and are
therefore guaranteed to be from the same domain.

32. The system of claim 30 wherein the set of tables includes a first table and a second table,
and wherein the definition defines that the first table relates to the second table by a many to one

relationship, and wherein the generating the set of tables includes automatically generating a foreign key column in the first table, wherein the foreign key column is for holding a foreign key to the second table.

33. The system of claim 30 wherein the set of tables includes a first table and a second table, and wherein the definition defines that the first table relates to the second table by a many to many relationship, and wherein the generating the set of tables includes automatically generating an associative table corresponding to the first table and the second table, and wherein the associative table has a unique value created for each unique many-to-many relationship between the first table and the second table.

34. The system of claim 30 wherein the set of tables includes a first table and a second table, and wherein the first table includes one or more columns from the second table, and wherein said one or more columns are automatically populated from the one or more columns.

35. The system of claim 34 wherein the definition of the system further includes a definition of the aggregates for the system, the system further comprising:

means for generating a set of aggregates from the definition of the aggregates and the definition of the schema.

36. The system of claim 33 wherein the definition of the system further includes a definition of the a user interface for the system, the system further comprising:

means for generating the user interface from the definition of the user interface and the definition of the schema.

37. The system of claim 33 wherein the definition of the system includes a definition of aggregates for use in the system and a definition of a query and reporting mechanism interface
5 for the system, the set of tables includes a set of fact tables and a set of dimension tables, and wherein the system further comprises:

means for generating the set of fact tables;

means for generating the set of dimension tables;

means for generating a set of aggregate tables; and

10 means for generating a query and reporting mechanism interface.

38. A computer program product comprising:

a memory medium; and

a computer program stored on the memory medium, the computer program comprising

instructions for accessing a definition of a system, the definition defining a schema for

15 use by the system, the schema defining a set of tables, a set of columns corresponding to

the set of tables, and a set of relationships between the tables of the set of tables, the

definition further defining a set of operations for manipulating the data, the set of

operations defining programs that operate on the set of tables and the set of table

columns, and instructions for using the definition to generate the set of tables.

39 The computer program product of claim 38 wherein the set of tables includes a first table and a second table, wherein the first table includes a first column, wherein the second table includes a second column, and wherein the first column and the second column are related by a join and are therefore guaranteed to be from the same domain.

5 40. The computer program product of claim 38 wherein the set of tables includes a first table and a second table, and wherein the definition defines that the first table relates to the second table by a many to one relationship, and wherein the generating the set of tables includes automatically generating a foreign key column in the first table, wherein the foreign key column is for holding a foreign key to the second table.

10 41. The computer program product of claim 38 wherein the set of tables includes a first table and a second table, and wherein the definition defines that the first table relates to the second table by a many to many relationship, and wherein the generating the set of tables includes automatically generating an associative table corresponding to the first table and the second table, and wherein the associative table has a unique value created for each unique many-to-many
15 relationship between the first table and the second table.

42. The computer program product of claim 38 wherein the set of tables includes a first table and a second table, and wherein the first table includes one or more columns from the second table, and wherein said one or more columns are automatically populated from the one or more columns.

43. A computer data signal embodied in a carrier wave comprising:

a computer program, the computer program comprising instructions for accessing a definition of a system, the definition defining a schema for use by the system, the schema defining a set of tables, a set of columns corresponding to the set of tables, and a set of relationships between the tables of the set of tables, the definition further defining a set of operations for manipulating the data, the set of operations defining programs that operate on the set of tables and the set of table columns, and instructions for using the definition to generate the set of tables.

44. The computer data signal embodied in the carrier wave of claim 43 wherein the set of tables includes a first table and a second table, wherein the first table includes a first column, wherein the second table includes a second column, and wherein the first column and the second column are related by a join and are therefore guaranteed to be from the same domain.

45. The computer data signal embodied in the carrier wave of claim 43 wherein the set of tables includes a first table and a second table, and wherein the definition defines that the first table relates to the second table by a many to one relationship, and wherein the generating the set of tables includes automatically generating a foreign key column in the first table, wherein the foreign key column is for holding a foreign key to the second table.

46. The computer data signal embodied in the carrier wave of claim 43 wherein the set of tables includes a first table and a second table, and wherein the definition defines that the first

table relates to the second table by a many to many relationship, and wherein the generating the set of tables includes automatically generating an associative table corresponding to the first table and the second table, and wherein the associative table has a unique value created for each unique many-to-many relationship between the first table and the second table.

- 5 47. The computer data signal embodied in the carrier wave of claim 43 wherein the set of tables includes a first table and a second table, and wherein the first table includes one or more columns from the second table, and wherein said one or more columns are automatically populated from the one or more columns.

The Abstract

A method of defining a well-formed database system by defining the organization of the data in the database, and by defining the operations for that data, is described. The definition can be used to automatically create and populate the well-formed database system. The well-formed database system conforms to rules of correctness and produces results that conform to the rules. The organization is defined by a data organization definition that specifies tables, their columns, and the relationships between tables. The operations define procedures that operate on the tables and the table columns. Importantly, the operations are defined along with the tables, columns, and relationships, so that the resulting system is well-formed.

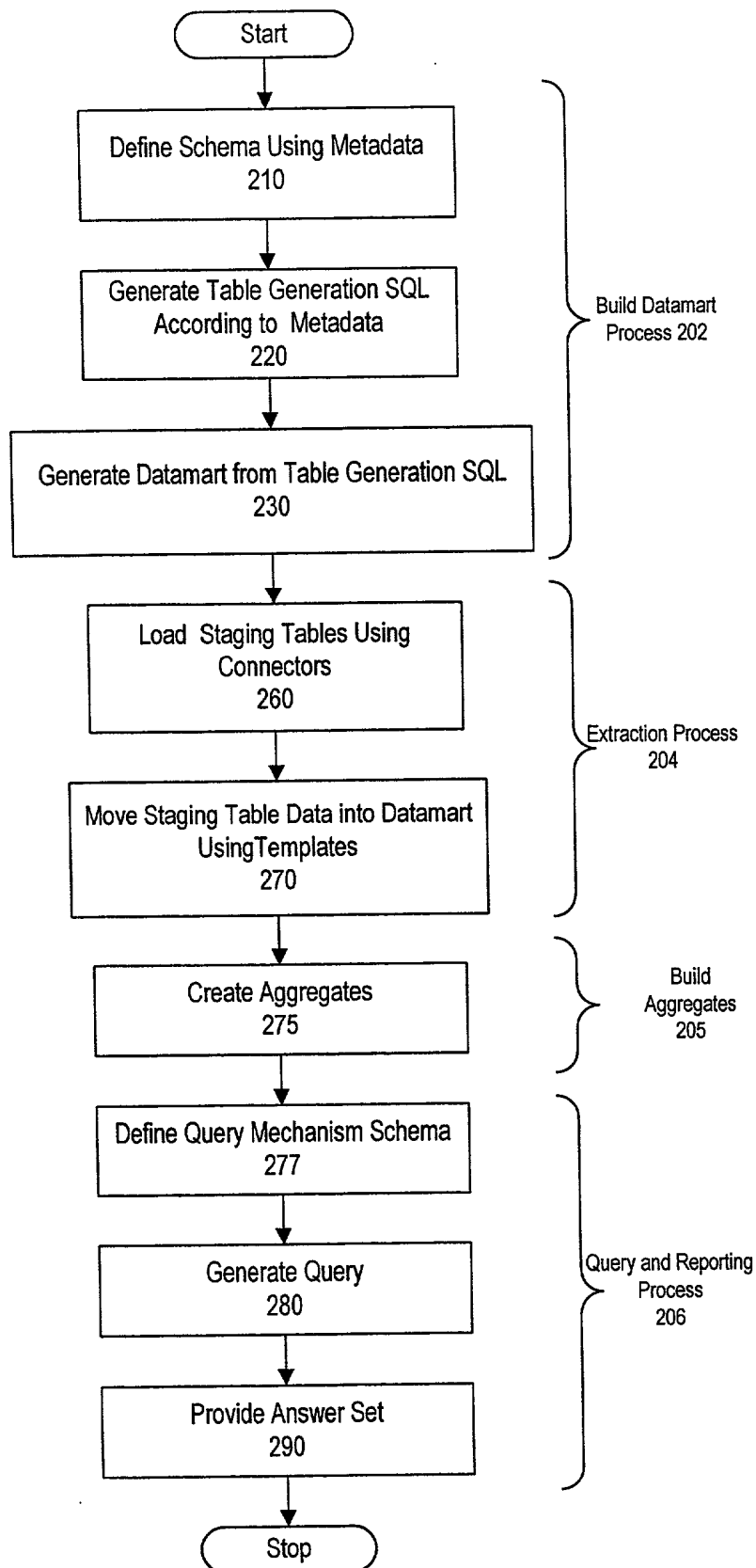


Figure 2

WOOD

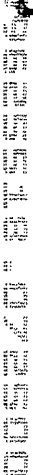


FIGURE 3

003

Date_0 560

[illegible]

bus_process

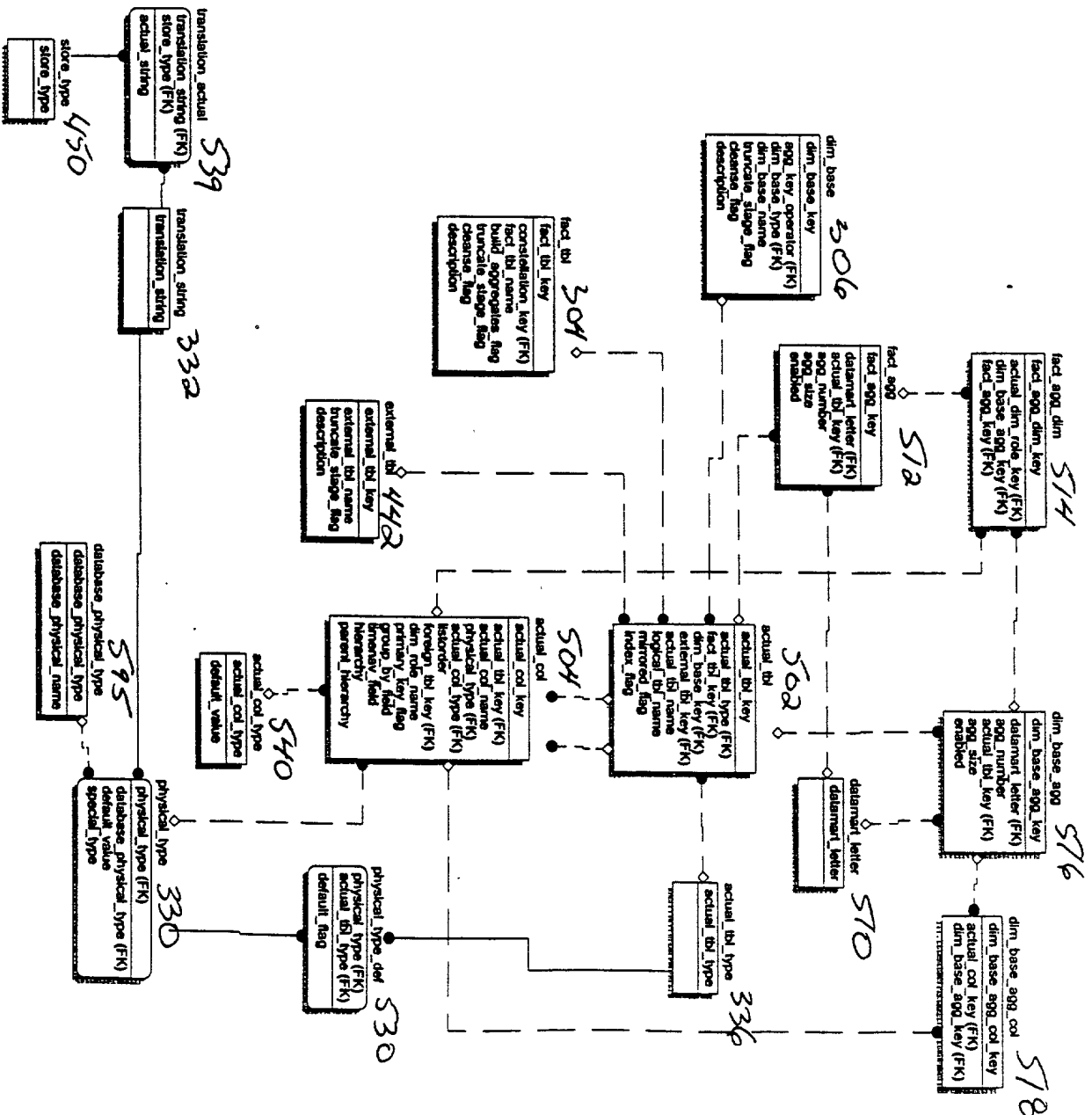
process_key	process_name
-------------	--------------

580

token_name	number_rows	listorder
------------	-------------	-----------

590

trans_type_0	trans_type_key	name	description
--------------	----------------	------	-------------



[illegible]

H:\PRIVATE\CLIENT\20308 Epiphany\701 MetaSchemas - EpiCenter System\Figure07 salesconstellation.vsd

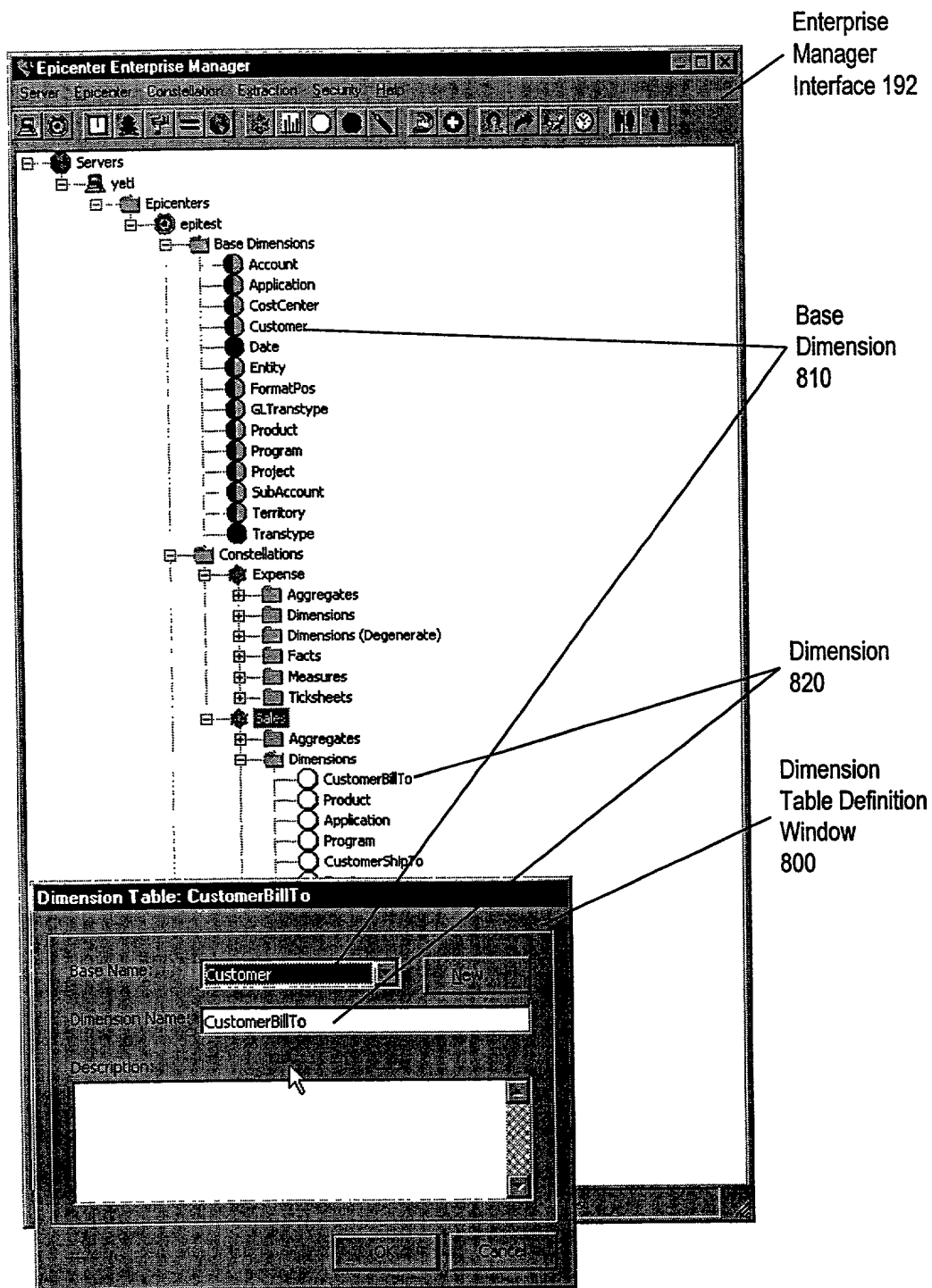


Figure 8

Dimension
Columns
920

Figure 9

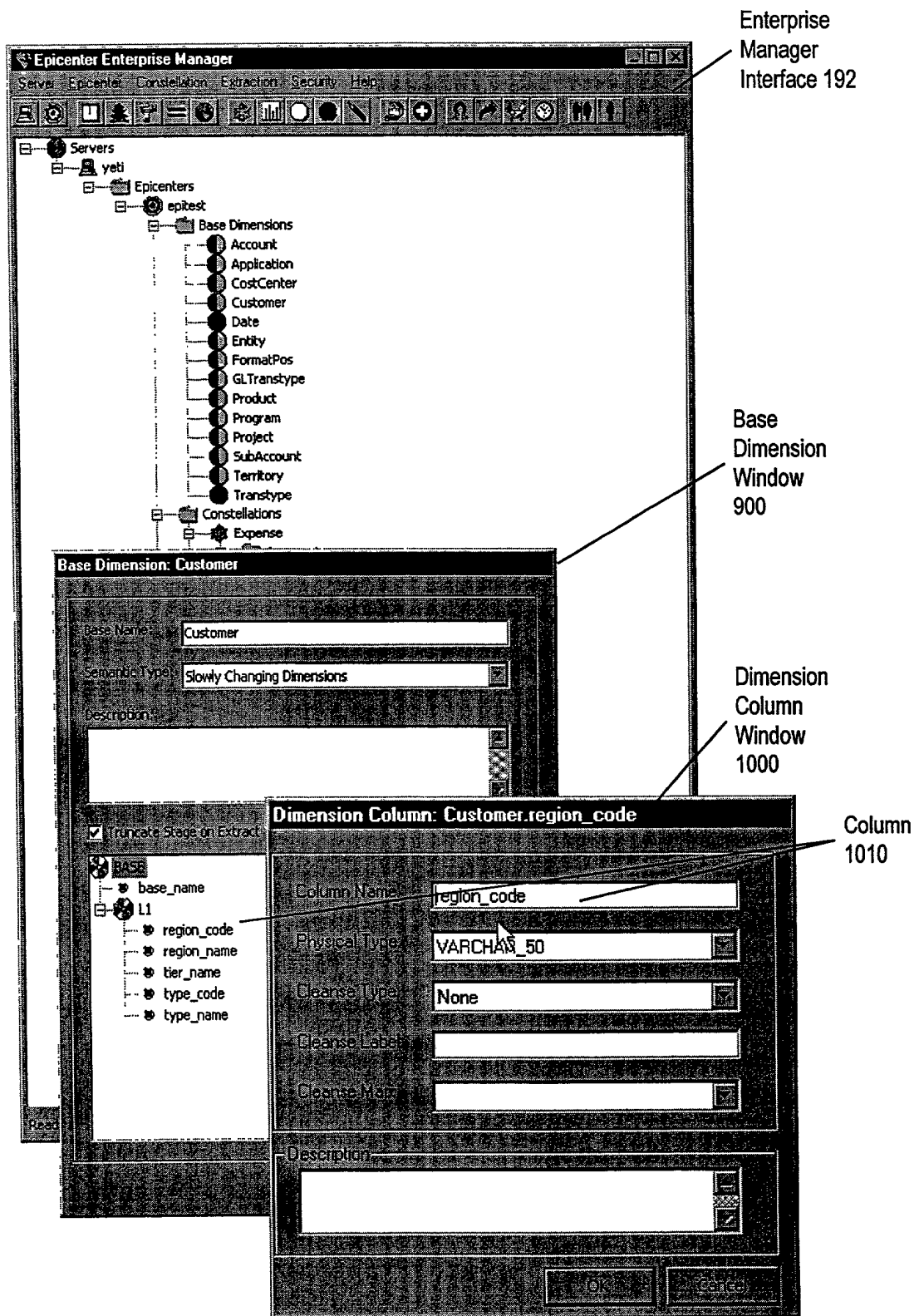
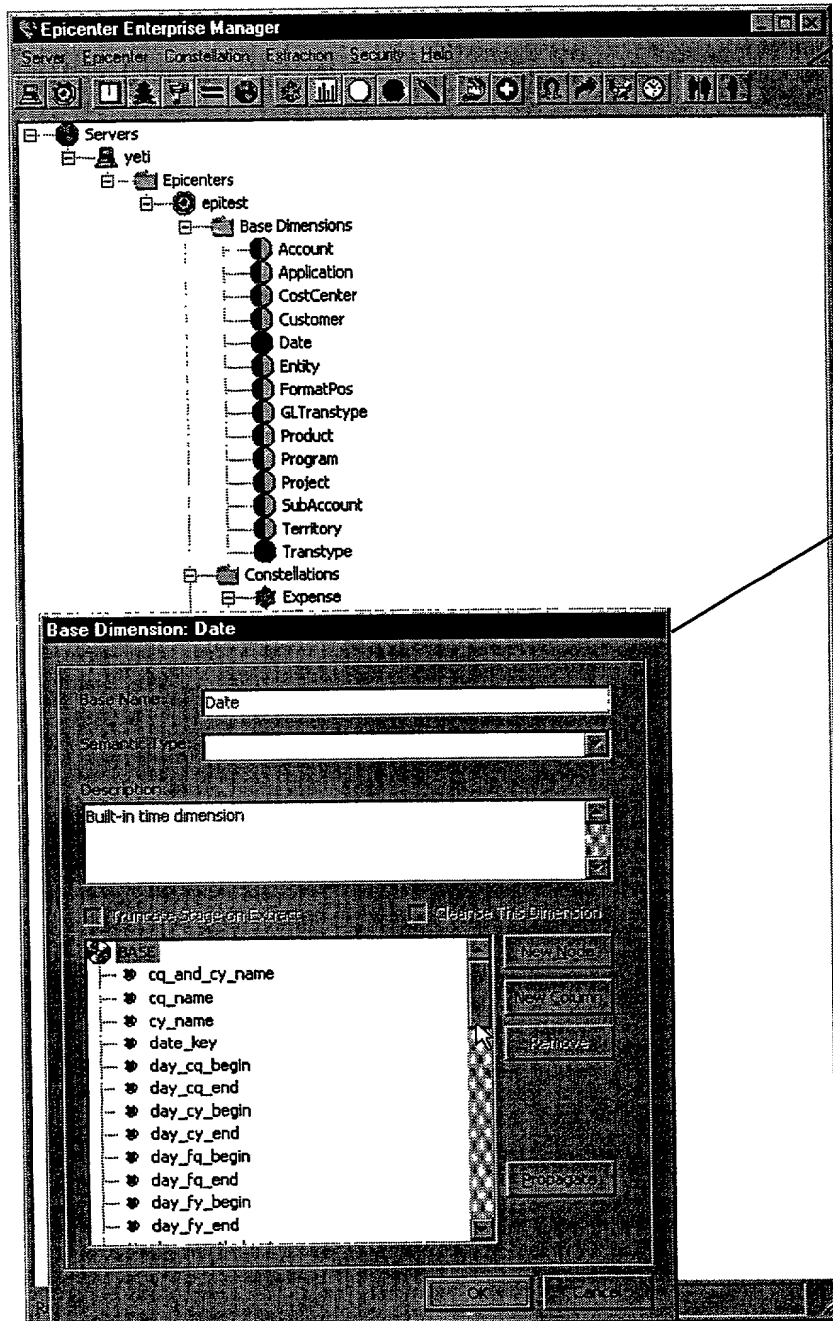


Figure 10



Enterprise
Manager
Interface 192

Base
Dimension
Window
900

Figure 11

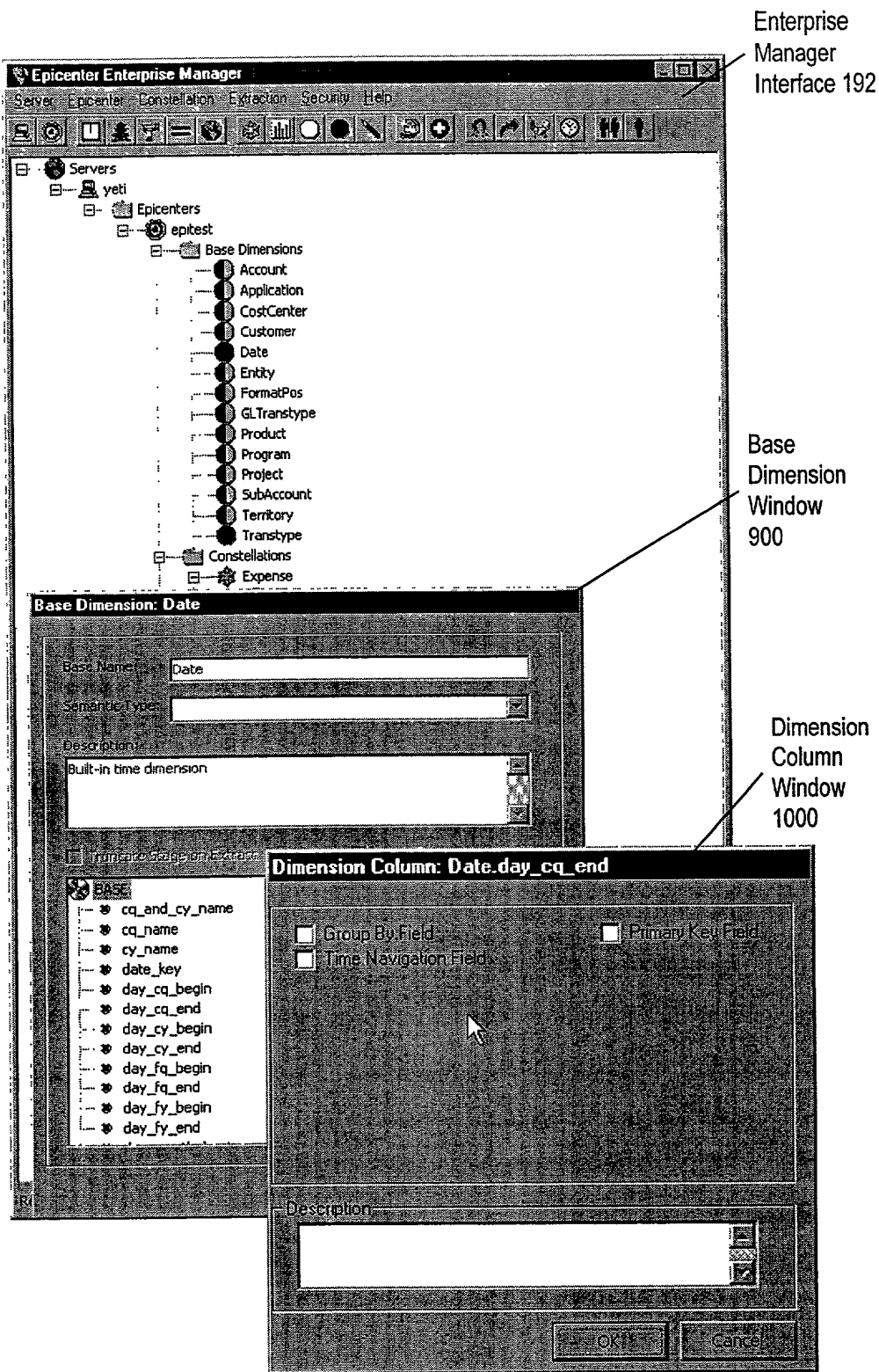


Figure 12

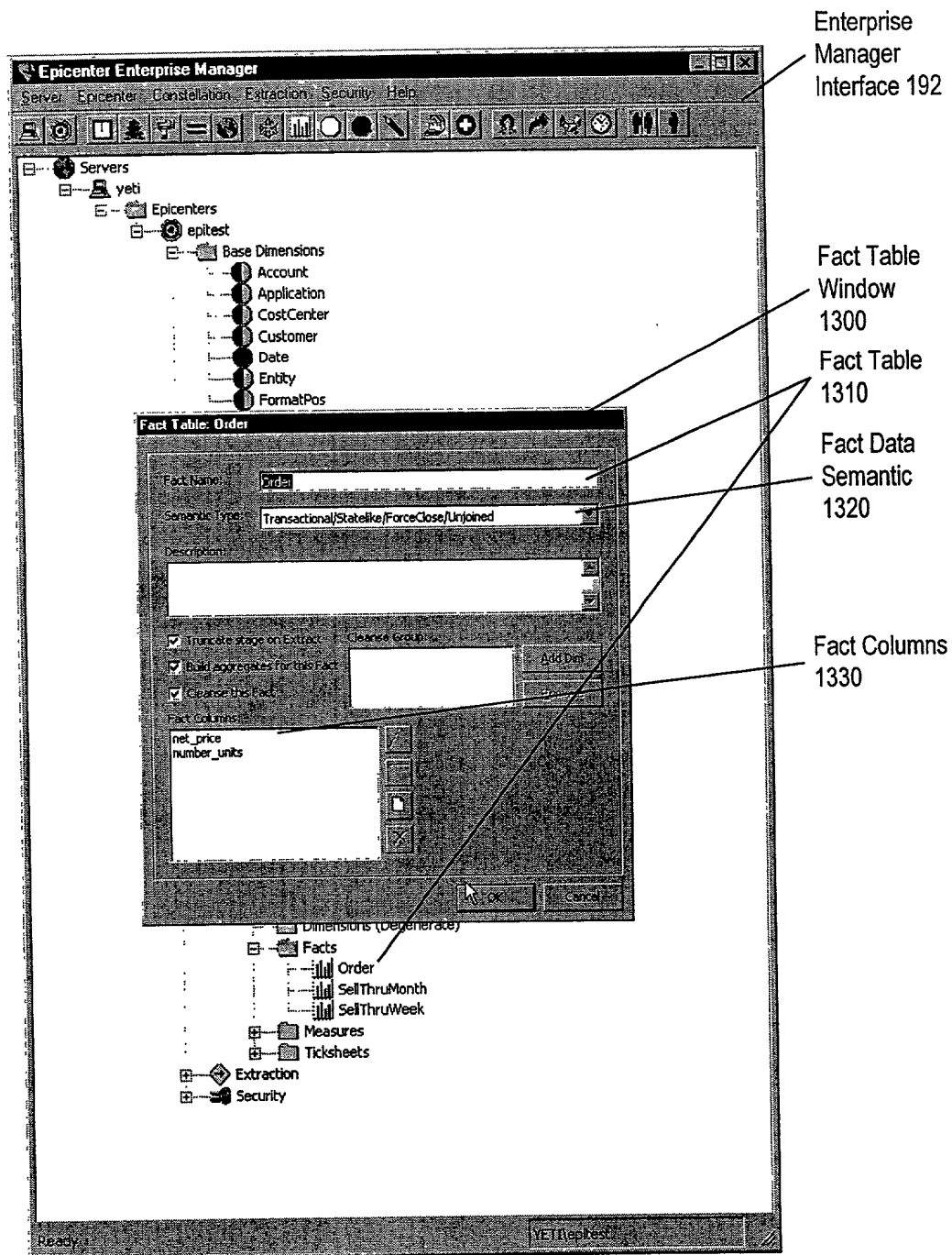


Figure 13

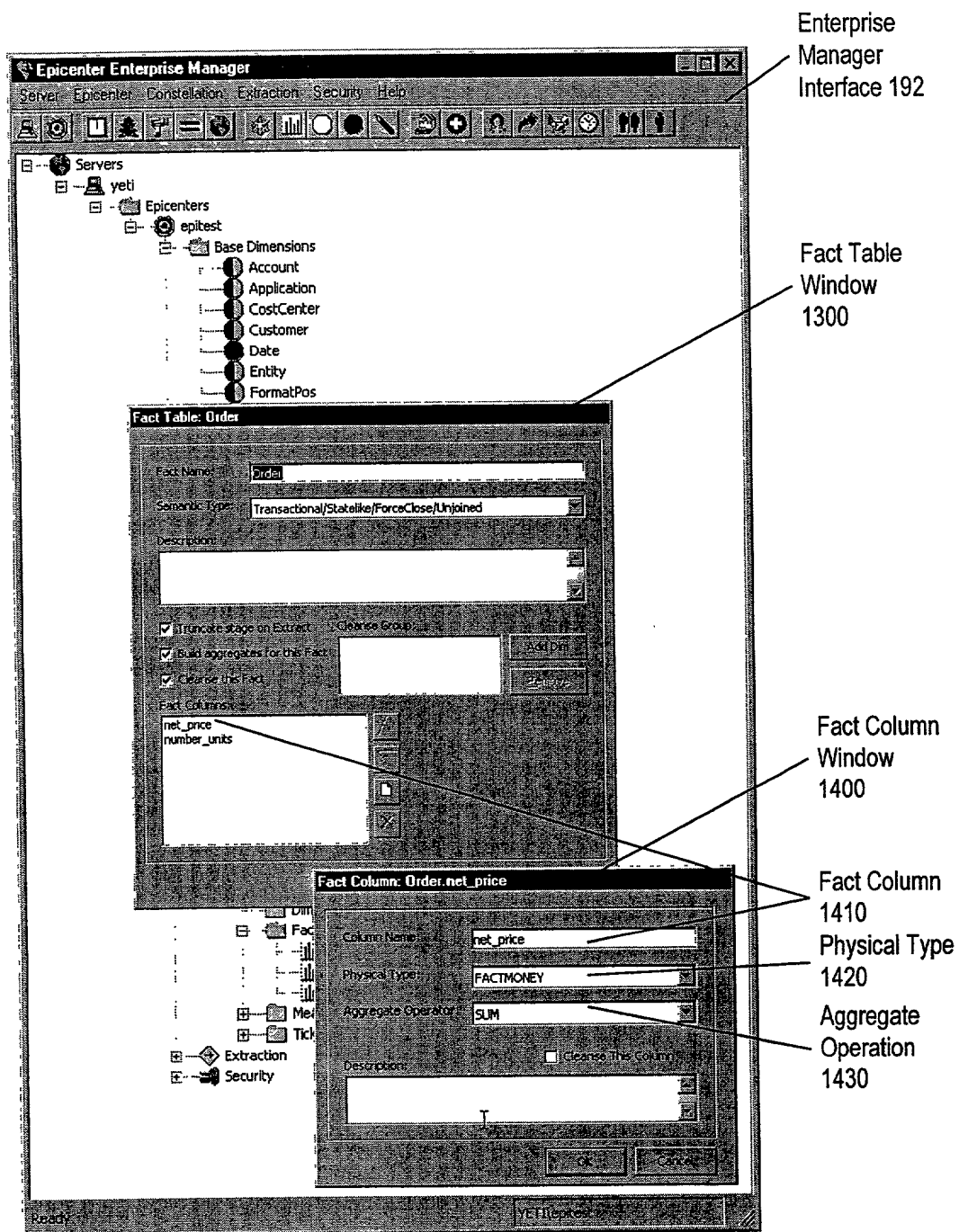


Figure 14

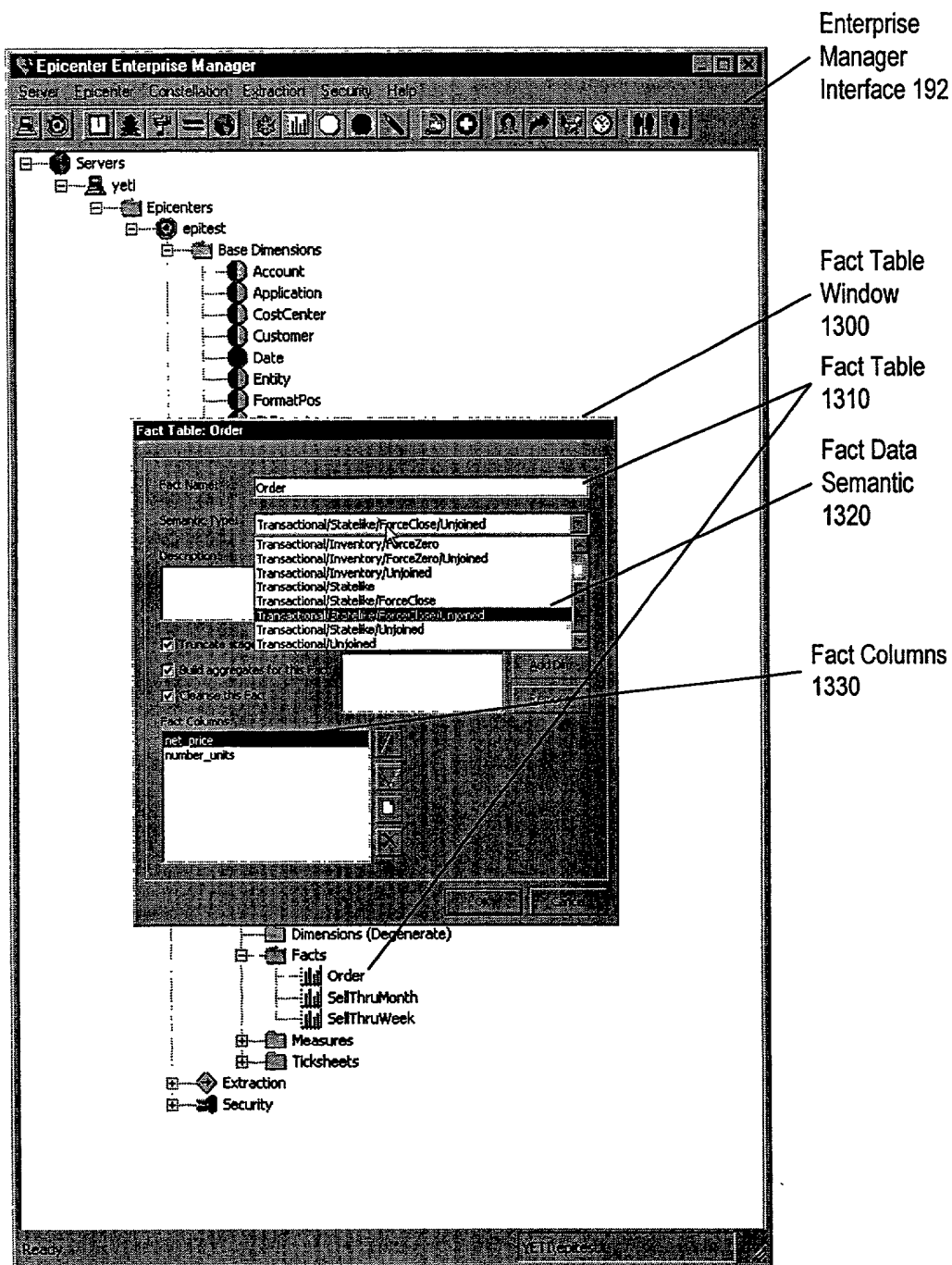


Figure 15

The screenshot shows the Epicenter Enterprise Manager interface. The top menu bar includes 'Server', 'Epicenter', 'Constellation', 'Extraction', 'Security', and 'Help'. The main window displays a tree view of the data source hierarchy:

- Servers
 - yeti
 - Epicenters
 - epitest
 - Base Dimensions
 - Account
 - Application
 - CostCenter
 - Customer
 - Date
 - Entity
 - FormatPos
 - GLTranstype
 - Product
 - Program
 - Project
 - SubAccount
 - Territory
 - Transtype
 - Constellations
 - Expense
 - Aggregates
 - Dimensions
 - Dimensions (Degenerate)
 - Facts
 - Measures
 - Ticketsheets
 - Sales
 - Aggregates

A dialog box titled 'Success Operation: Generating Datamart Schema' is open in the foreground. It contains a 'Description' section with the text: 'Successfully generated the following objects:'. Below this, a list of generated objects is displayed:

- Created: Account_0_A
- Created: Account_0_B
- Created: AccountStage
- Created: AccountMap_A
- Created: AccountMap_B
- Created: Application_0_A
- Created: Application_0_B
- Created: ApplicationStage
- Created: ApplicationMap_A
- Created: ApplicationMap_B

The dialog box has a 'Yes' button at the bottom right.

Batch
Operation
Window
1600

H:\PRIVATE\CLIENT\20308 Epiphany\701 MetaSchemas - EpiCenter System\Figure16 generatingschema.vsd

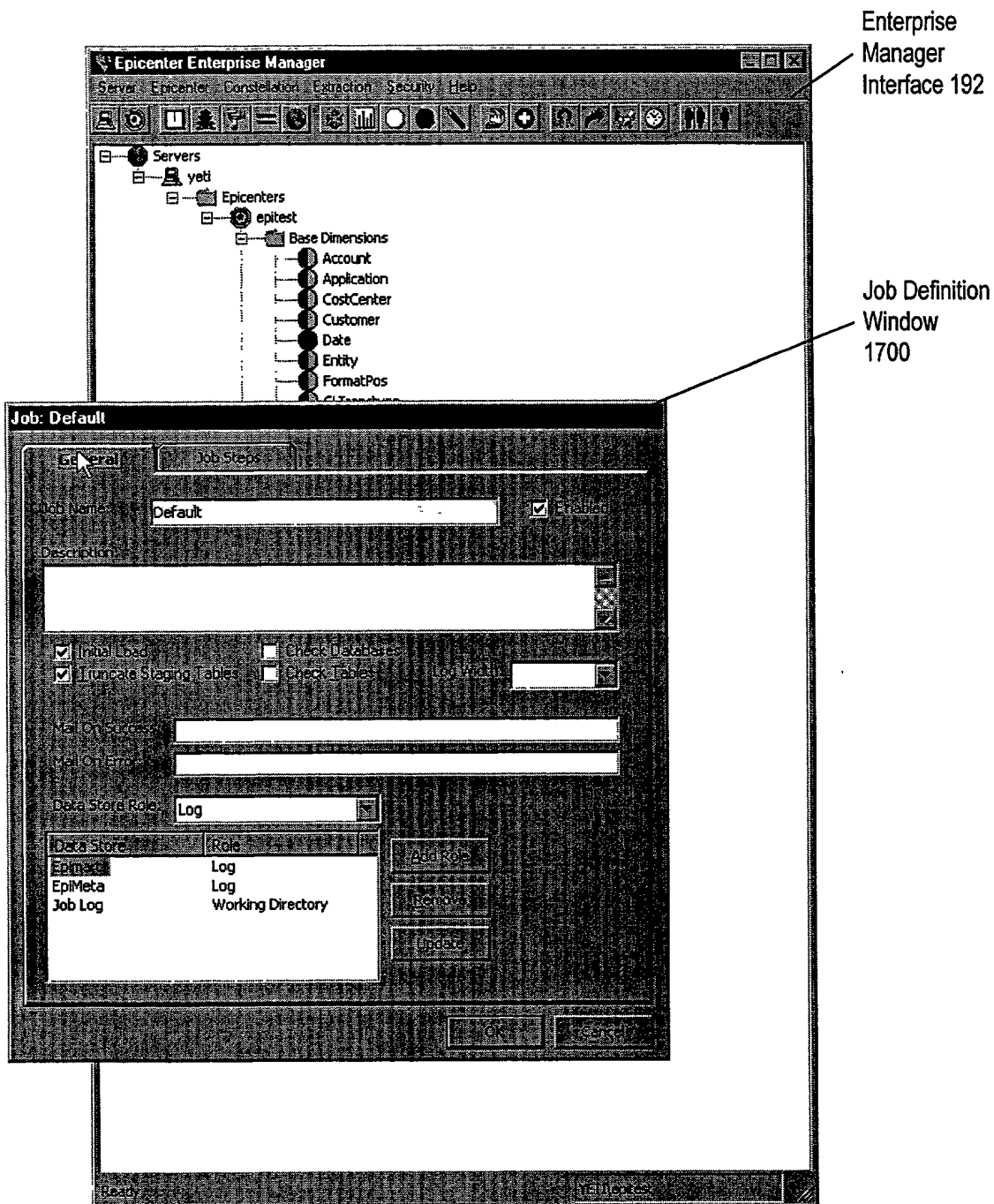


Figure 17

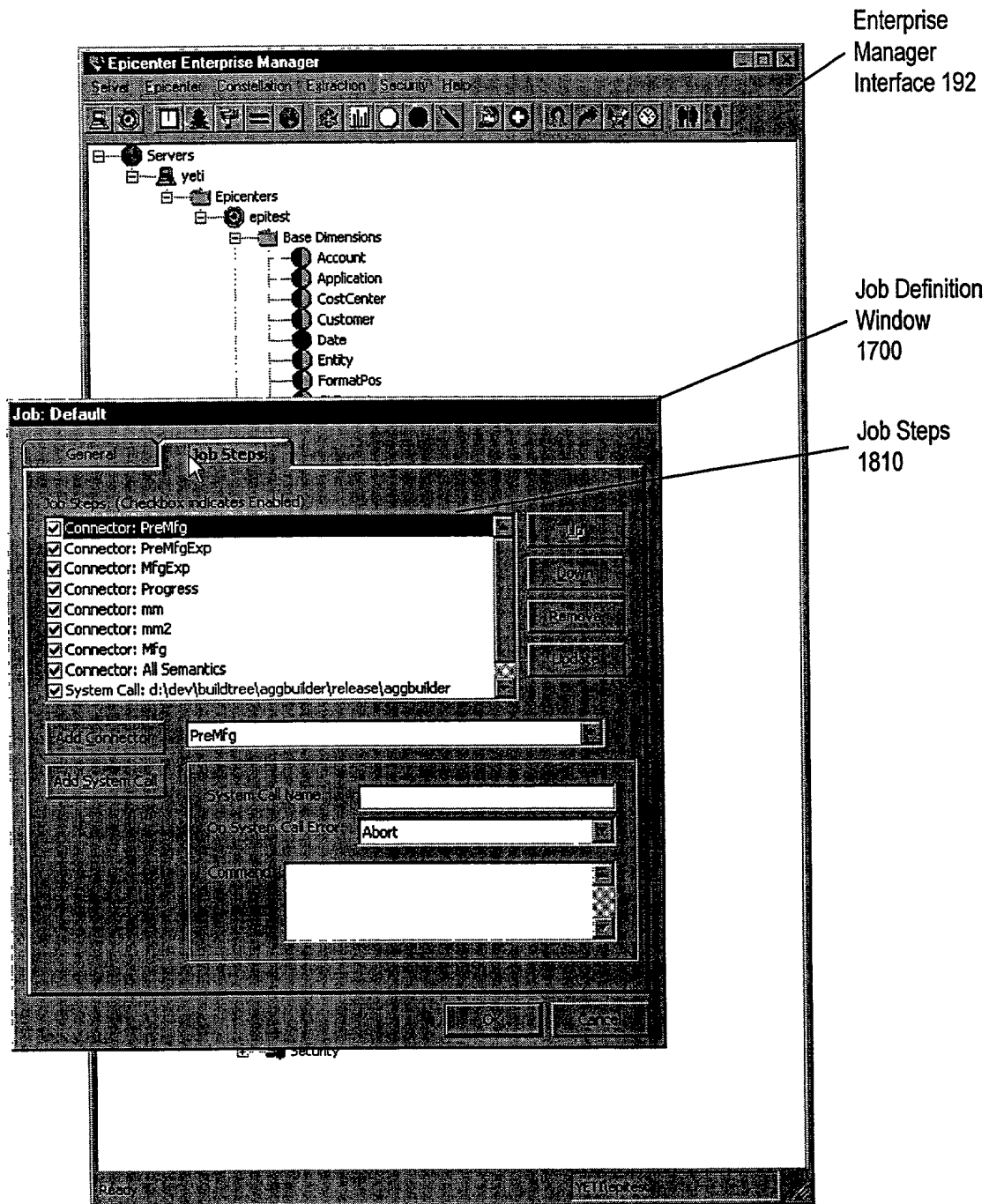


Figure 18

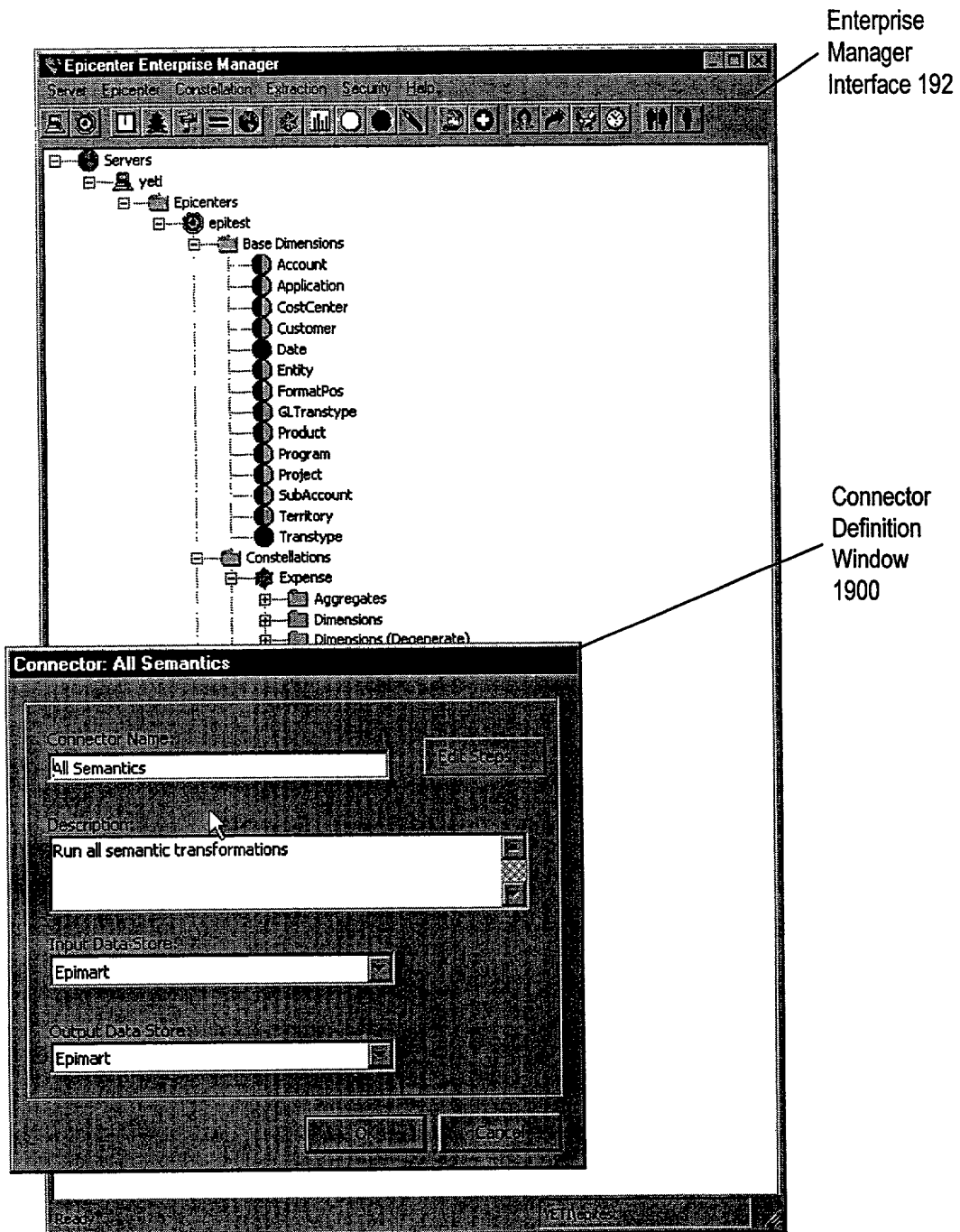


Figure 19

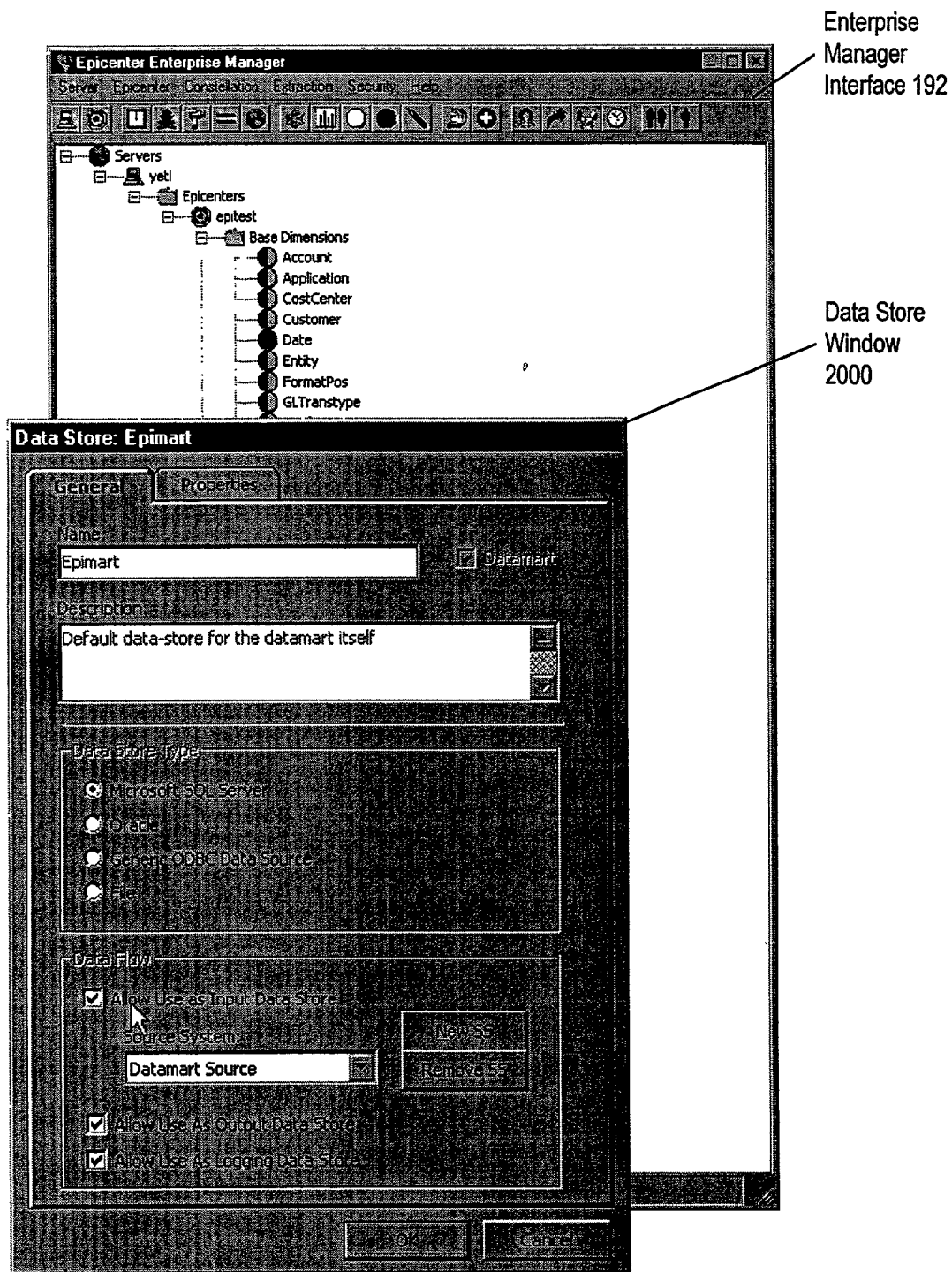


Figure 20

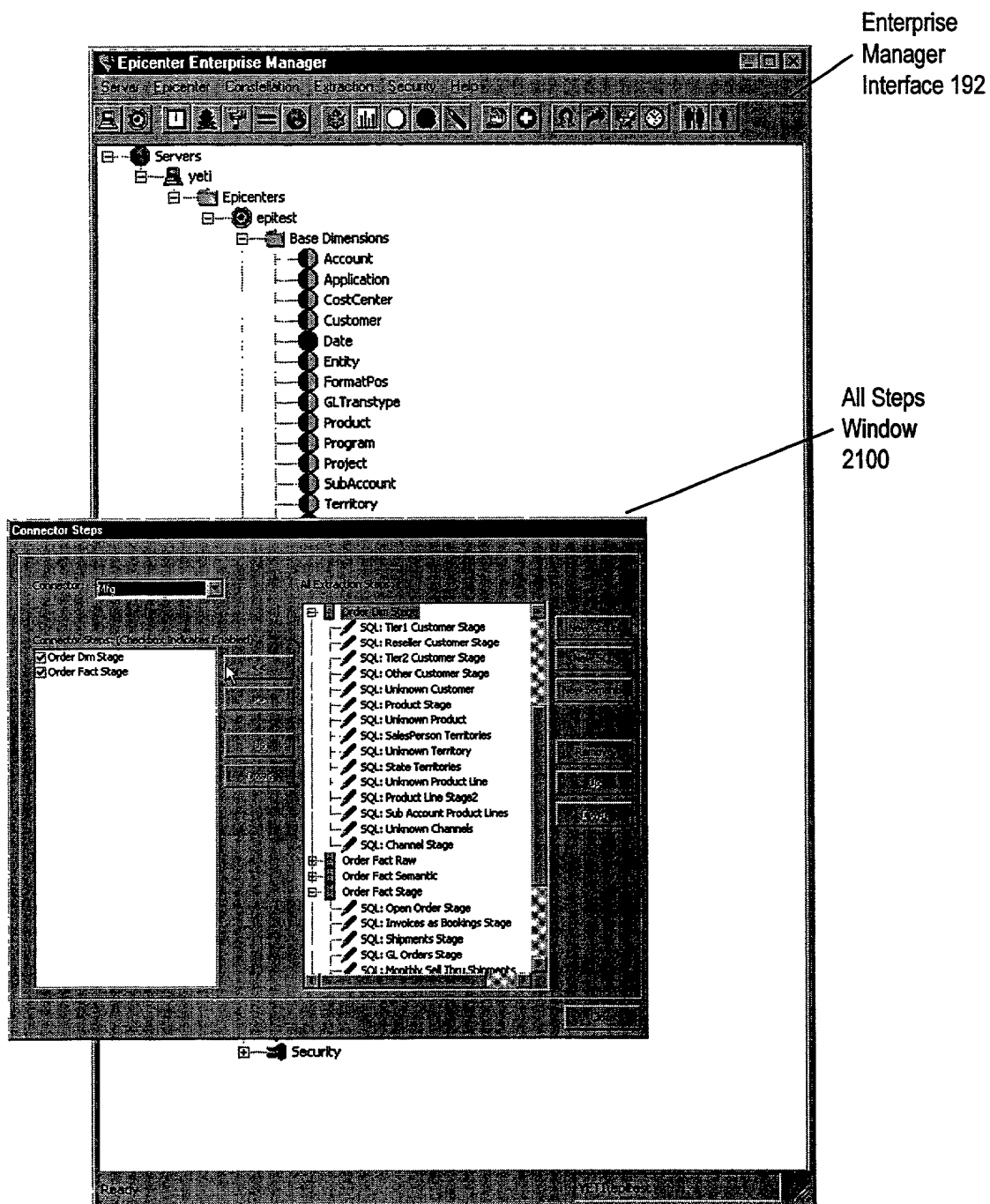


Figure 21

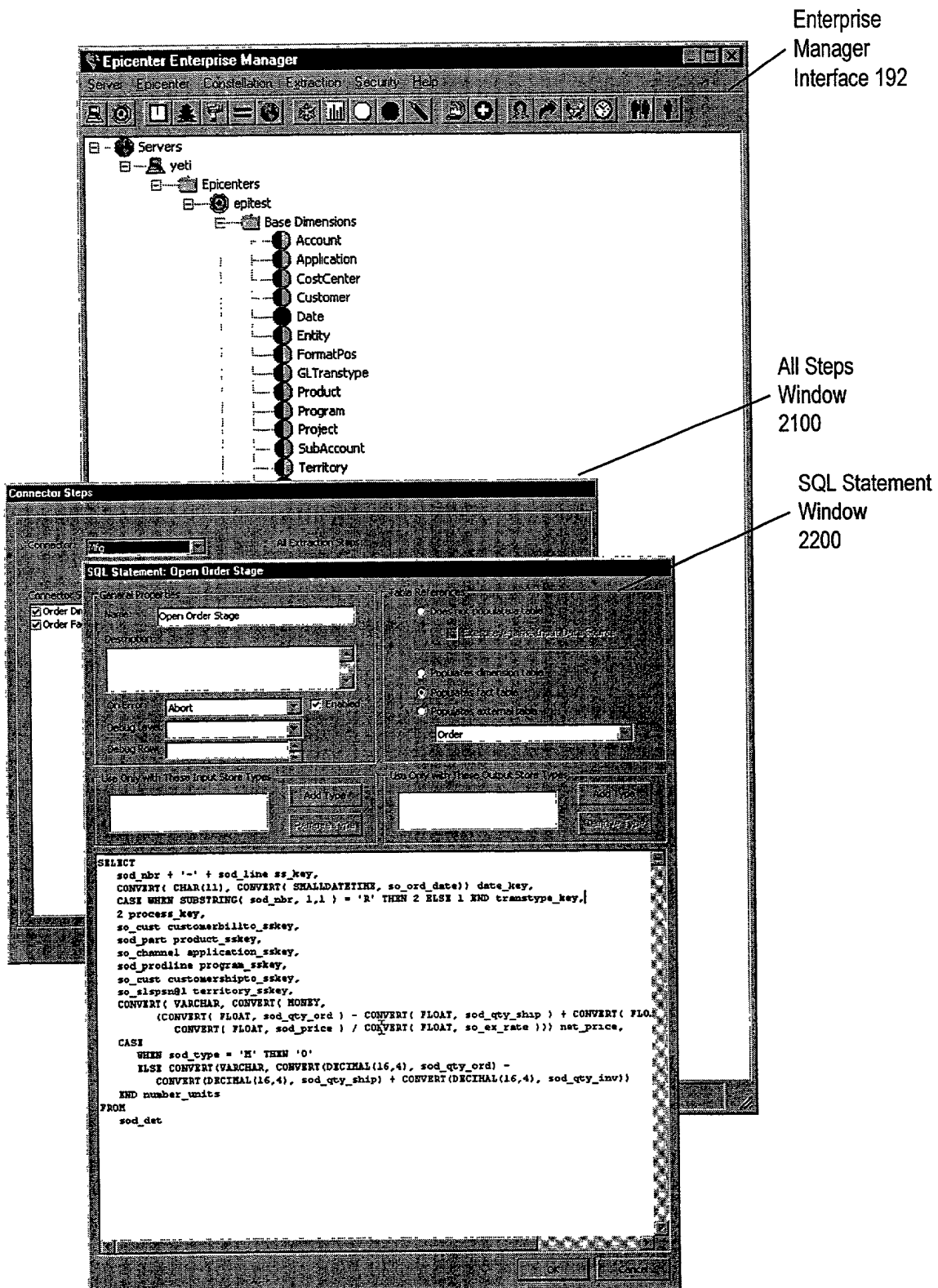


Figure 23

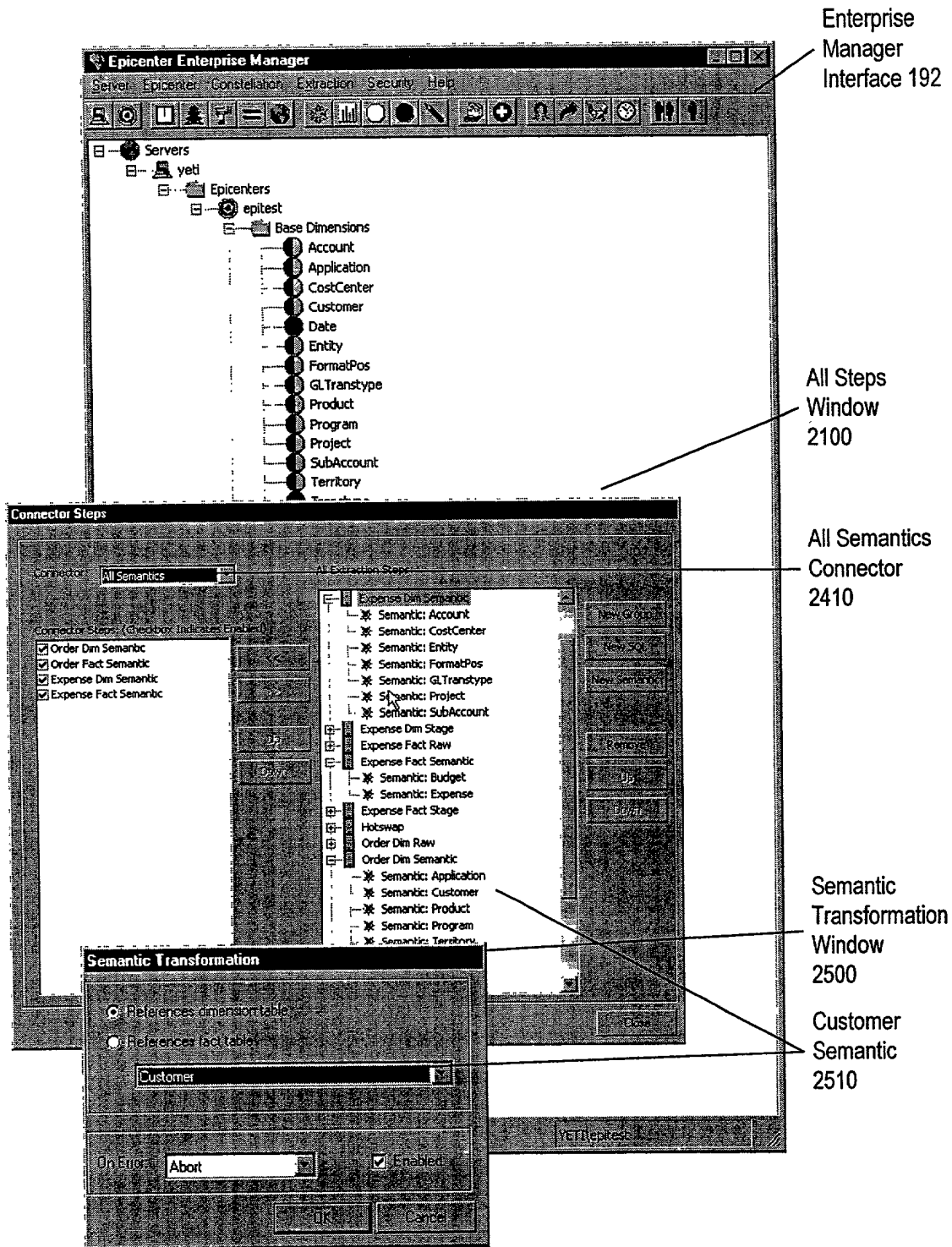


Figure 25

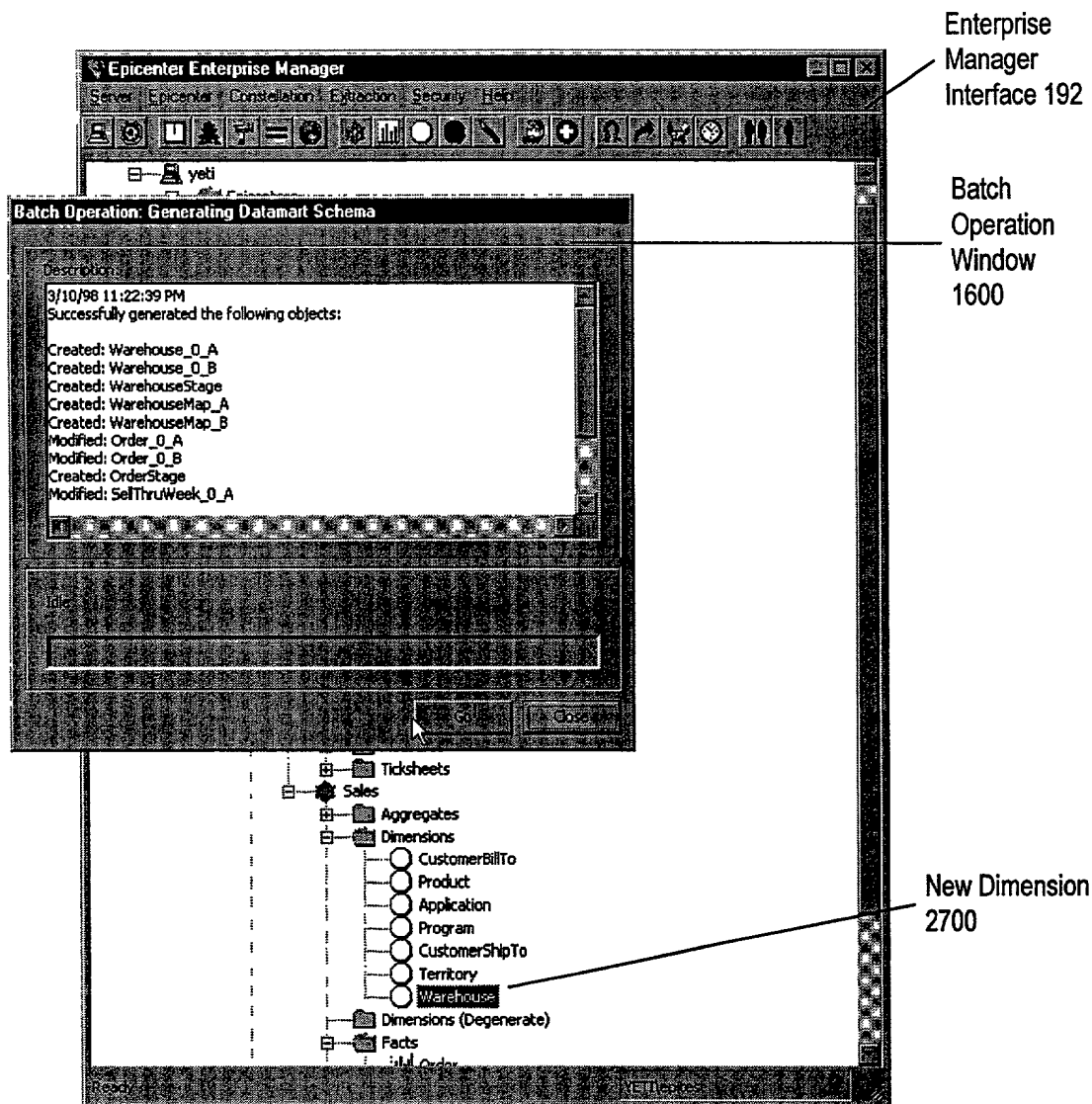


Figure 27

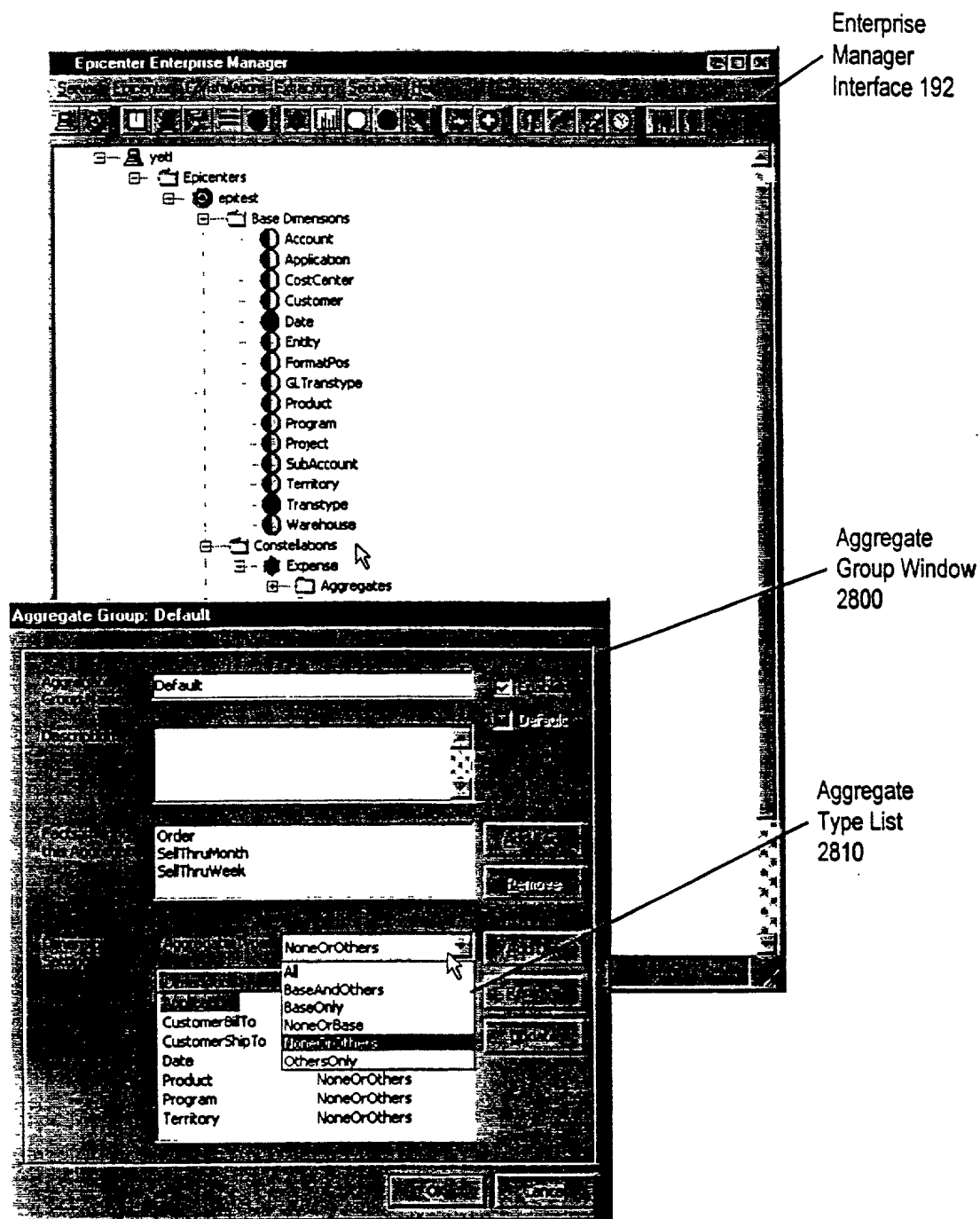
[illegible]

Figure 28

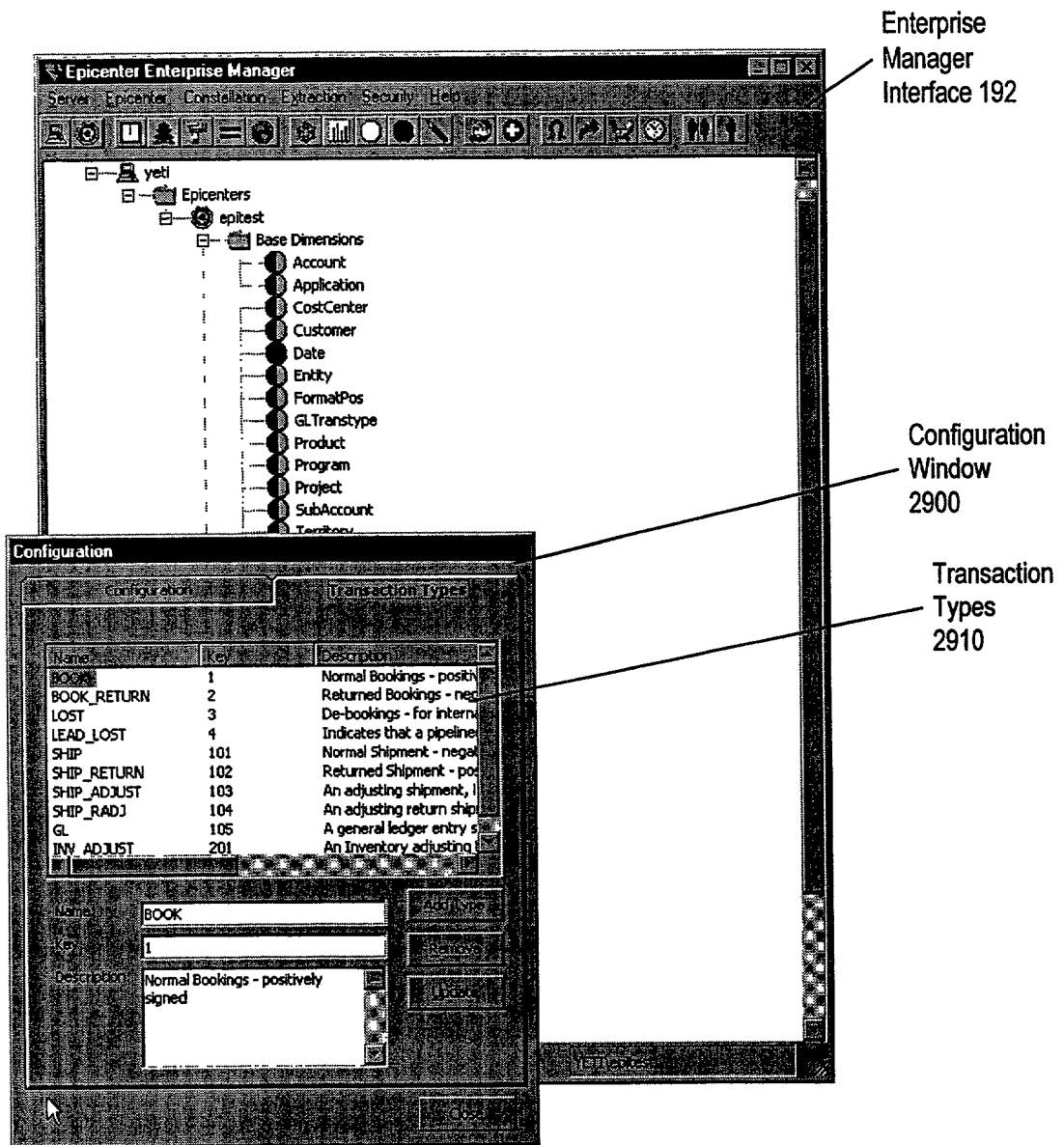


Figure 29

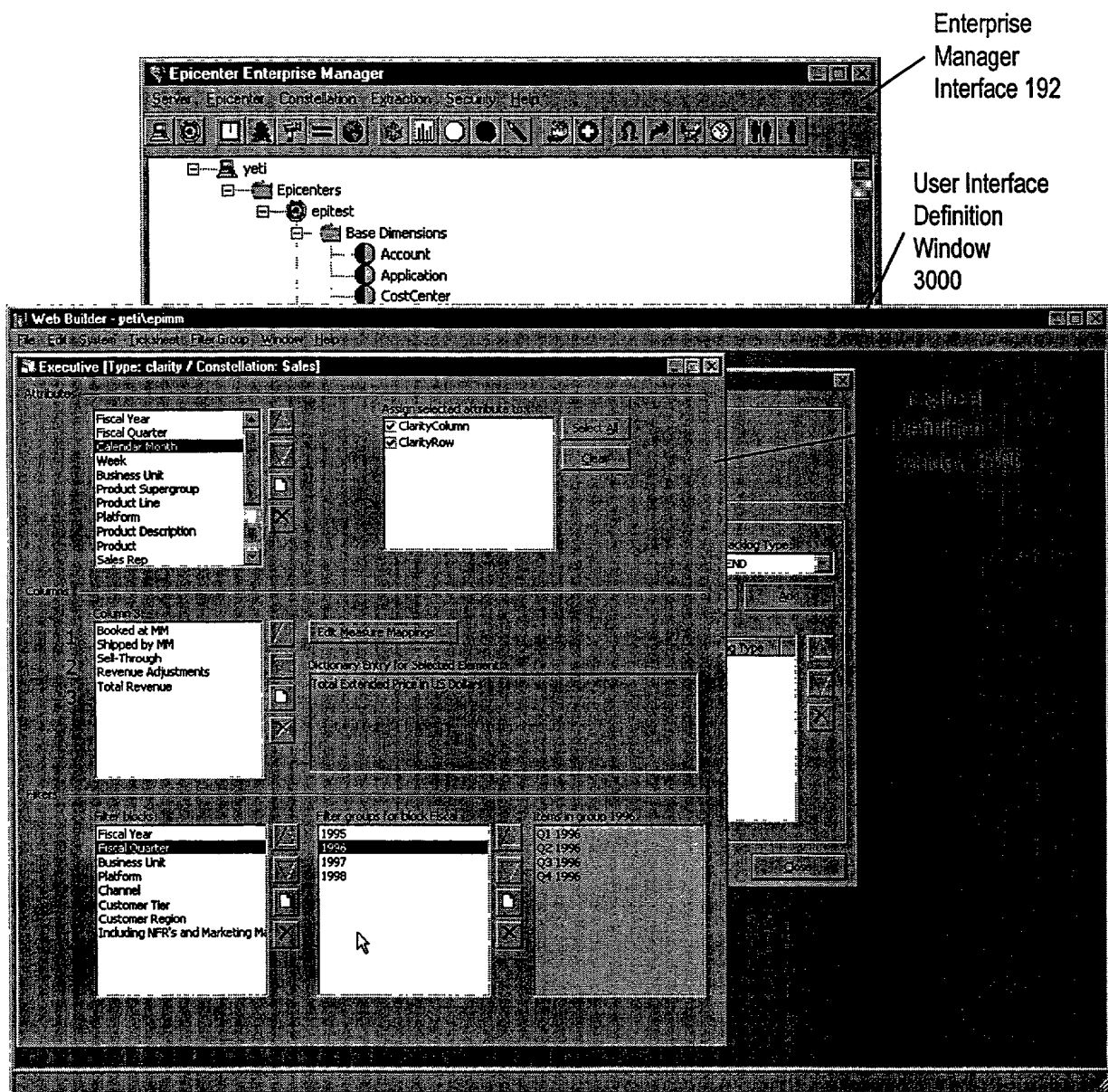


Figure 31

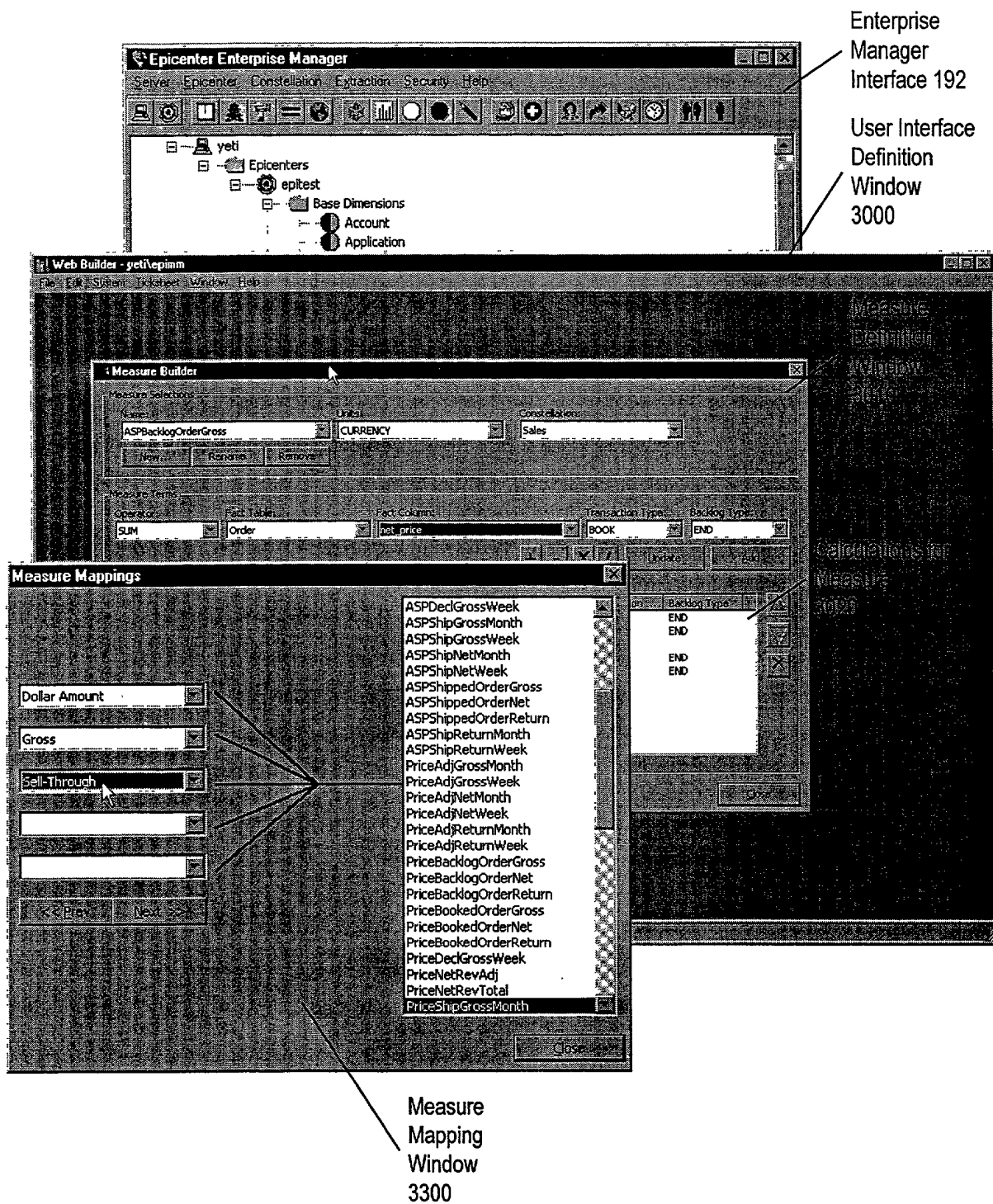


Figure 33

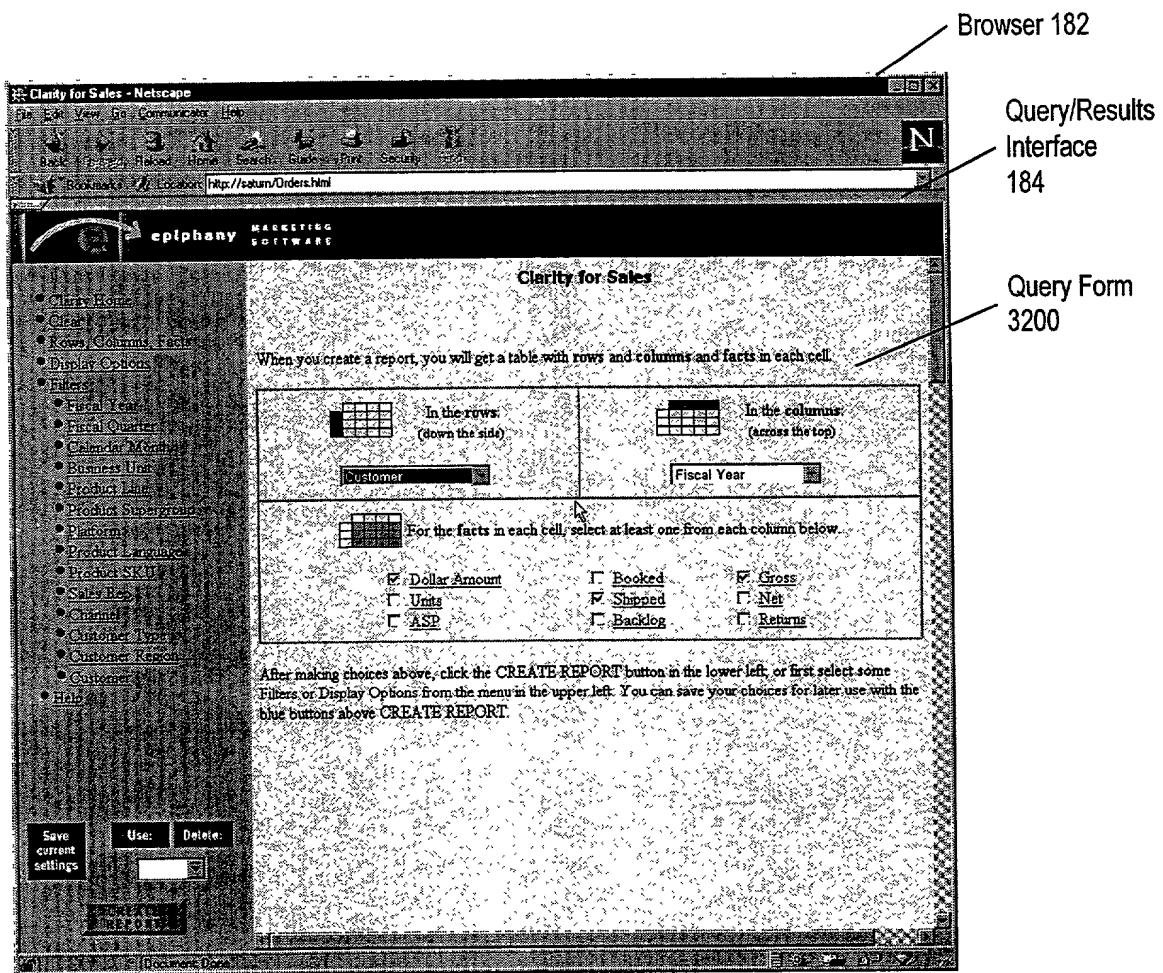


Figure 34

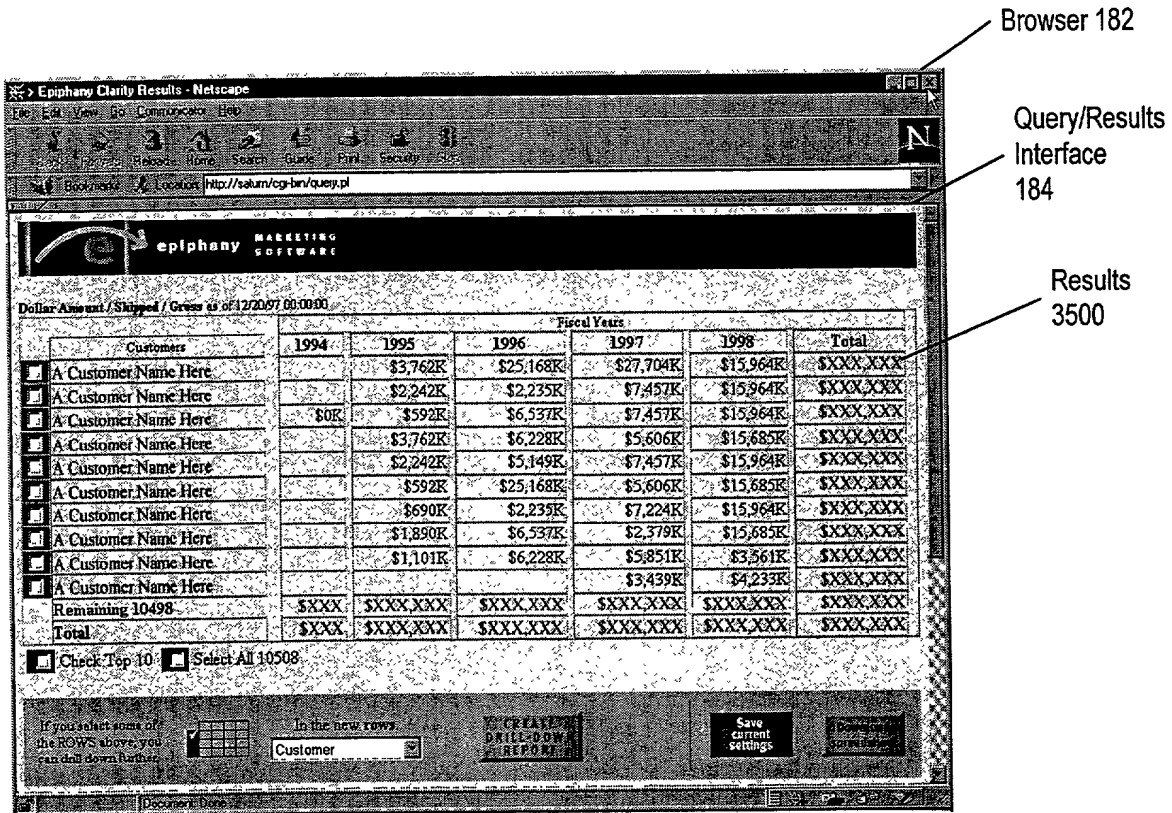


Figure 35

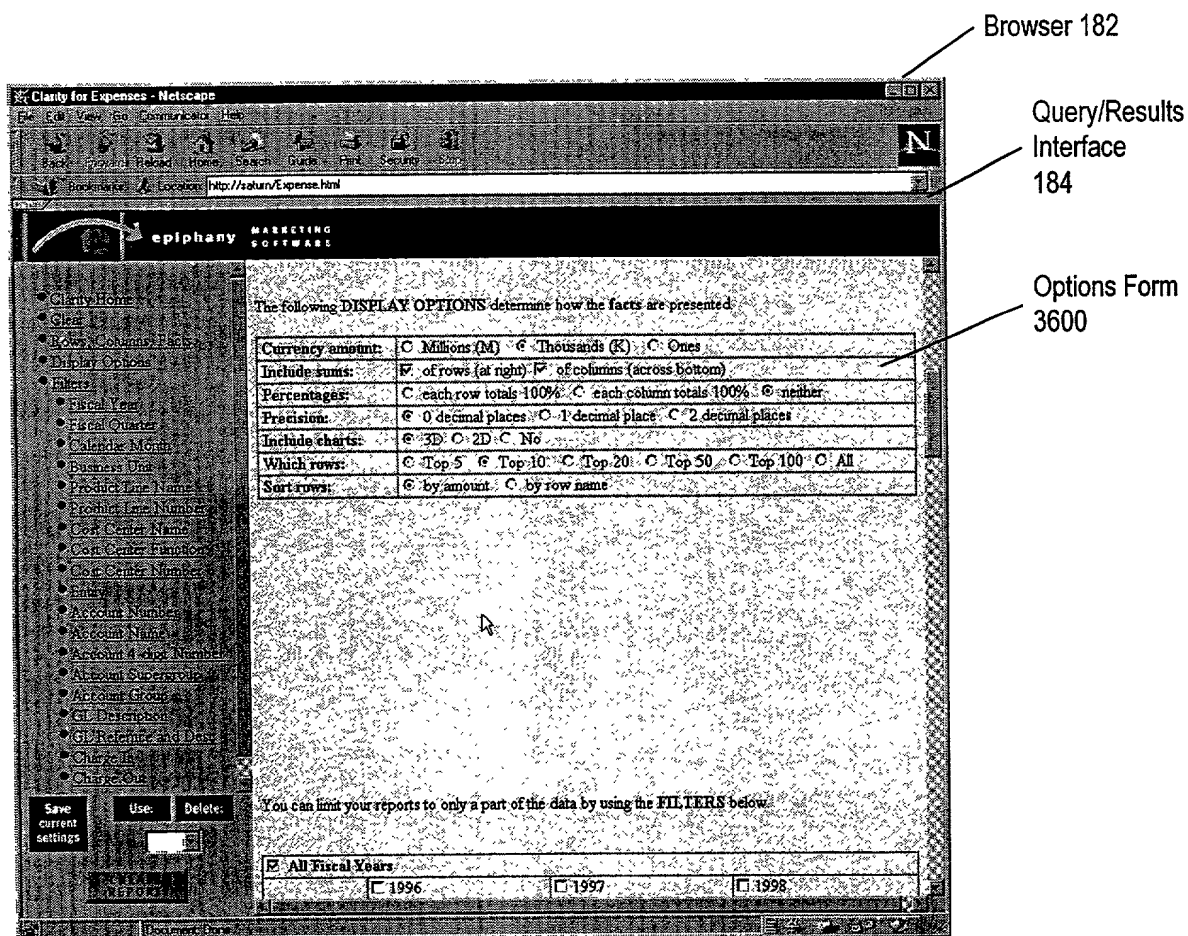


Figure 36

Please type a plus sign (+) inside this box



PTO/SB/01 (12-97)

Approved for use through 9/30/00.OMB 0651-0032

Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION FOR UTILITY OR DESIGN PATENT APPLICATION (37 CFR 1.63)	Attorney Docket Number	20308-713
	First Named Inventor	Craig D. Weissman
	<i>COMPLETE IF KNOWN</i>	
	Application Number	Not Yet Assigned
	Filing Date	Herewith
	Group Art Unit	Not Yet Assigned
<input checked="" type="checkbox"/> Declaration Submitted with Initial Filing	OR	<input type="checkbox"/> Declaration Submitted after Initial Filing (surcharge (37 CFR 1.16(e)) required)
Examiner Name		Not Yet Assigned

As a below named Inventor, I hereby declare that:

My residence, post office address, and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

**METHOD AND APPARATUS FOR CREATING A WELL-FORMED DATABASE
SYSTEM USING A COMPUTER**

(Title of the Invention)

the specification of which

☒ is attached hereto

OR

☐ was filed on (MM/DD/YYYY)

as United States Application Number or PCT International

Application Number and was amended on (MM/DD/YYYY) (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment specifically referred to above.

I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56.

I hereby claim foreign priority benefits under 35 U.S.C. 119(a)-(d) or 365(b) of any foreign application(s) for patent or inventor's certificate, or 365(a) of any PCT international application which designated at least one country other than the United States of America, listed below and have also identified below, by checking the box, any foreign application for patent or inventor's certificate, or of any PCT international application having a filing date before that of the application on which priority is claimed.

Prior Foreign Application Number(s)	Country	Foreign Filing Date (MM/DD/YYYY)	Priority Not Claimed	Certified Copy Attached?	
			<input type="checkbox"/>	YES	NO
			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
			<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

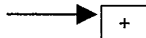
☐ Additional foreign application numbers are listed on a supplemental priority data sheet PTO/SB/028 attached hereto:

I hereby claim the benefit under 35 U.S.C. 119(h) of any United States provisional application(s) listed below.

Application Number(s)	Filing Date (MM/DD/YYYY)	<input type="checkbox"/> Additional provisional application numbers are listed on a supplemental priority data sheet PTO/SB/028 attached hereto.
60/145,700	07/26/99	

(Page 1 of 2)

Burden Hour Statement: This form is estimated to take 0.4 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Washington, DC 20231.

Please Type a plus sign (+) inside this box 

PTO/SB/01 (12-97)

Approved for use through 9/30/00.OMB 0651-0032

Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION — Utility or Design Patent Application


I hereby claim the benefit under 35 U.S.C. 120 of any United States application(s), or 365(c) of any PCT international application designating the United States of America, listed below and, insofar as the subject matter of each of the claims of the application is not designated in the prior United States or PCT international application in the manner provided by the first paragraph of 31 U.S.C. 112. I acknowledge the duty to disclose information which is material to patentability as defined in 37 CFR 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application.

U.S. Parent Application or PCT Parent Number

Parent Filing Date
(MM/DD/YYYY)

Parent Patent Number
(if applicable)

☐ Additional U.S. or PCT international application numbers are listed on a supplemental priority data sheet PTO/SB/028 attached hereto.

As a named inventor, I hereby appoint the following registered practitioner(s) to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith: ☒ Customer Number 21971 

OR

☐ Registered practitioner(s) name/registration number listed below

Place Customer
Number Bar
Code Label here

Name

Registration
Number

Name

Registration
Number

☐ Additional registered practitioner(s) named on supplemental Registered Practitioner Information sheet PTO/SB/02C attached hereto.

Direct all correspondence to: ☒ Customer Number
or Bar Code Label

21971

OR ☐ Correspondence address below

Name Richard L. Gregory, Jr.

Address Wilson Sonsini Goodrich & Rosati

Address 650 Page Mill Road

City Palo Alto

State

CA

ZIP

94304

Country U.S.

Telephone

650-493-9300

Fax

650-493-6811

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. 1001 and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Name of Sole or First Inventor:

☐ A petition has been filed for this unsigned inventor

Given Name (first and middle (if any))

Family Name or Surname

Craig David

Weissman

Inventor's Signature

Date

Residence: City

Belmont

State

CA.

Country

USA

Citizenship

USA

Post Office Address

735 Old Country Road, Apartment C

Post Office Address

City

Belmont

State

CA.


ZIP

94002

Country

USA

☒ Additional inventors are being named on the (One) 1 supplemental Additional Inventor(s) sheet(s) PTO/SB/02A attached hereto:

Please Type a plus sign (+) inside this box 

PTO/SB/02A (3-97)

Approved for use through 9/30/98, OMB 0651-0032

Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it contains a valid OMB control number.

DECLARATION	ADDITIONAL INVENTOR(S) Supplemental Sheet Page <u>1</u> of <u>1</u>
--------------------	---

Name of Additional Joint Inventor, if any:				<input type="checkbox"/> A petition has been filed for this unsigned inventor			
Given Name (first and middle (if any))				Family Name or Surname			
Gregory Vincent				Walsh			
Inventor's Signature				Date			
Residence: City	Cupertino	State	CA.	Country	USA	Citizenship	USA
Post Office Address		16000 Montebello Road					
Post Office Address							
City	Cupertino	State	CA.	ZIP	95014	Country	USA
Name of Additional Joint Inventor, if any:				<input type="checkbox"/> A petition has been filed for this unsigned inventor			
Given Name (first and middle (if any))				Family Name or Surname			
Eliot Leonard				Wegbreit			
Inventor's Signature				Date			
City	Palo Alto	State	CA.	Country	USA	Citizenship	USA
Post Office Address		1516 Dana Avenue					
Post Office Address							
City	Palo Alto	State	CA.	ZIP	94303	Country	USA
Name of Additional Joint Inventor, if any:				<input type="checkbox"/> A petition has been filed for this unsigned inventor			
Given Name (first and middle (if any))				Family Name or Surname			
Ankur S.				Jain			
Inventor's Signature				Date			
City	Sunnyvale	State	CA.	Country	USA	Citizenship	USA
Post Office Address		1063 Morse Avenue #22-301					
Post Office Address							
City	Sunnyvale	State	CA.	ZIP	94089	Country	India

Burden Hour Statement: This form is estimated to take 0.4 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Washington, DC 20231.

Appendix A

Appendix A illustrates semantic types that may be supported and their corresponding adaptive template names. For example, the Pipelined semantic type is made up of, in this order, the map_keys the pipe_state and the index_fact adaptive templates. The example pre-parsed and post parsed SQL adaptive templates are then provided.

As mentioned previously, the use of the semantic types significantly reduces the amount of work needed to implement the datamart 150. By selecting a semantic type for a particular fact table or dimension table, the consultant automatically selects the corresponding pre-parsed SQL adaptive templates. The selected adaptive templates are then automatically converted into post parsed SQL statements that include the schema specific information for the datamart 150.

Additionally, these post parsed SQL statements include the SQL for accessing and manipulating the datamart 150 tables.

semantic_type_name	adaptive_template_name
Pipelined	map_keys
Pipelined	pipe_state
Pipelined	index_fact
Pipelined/Unjoined	upd_unj
Pipelined/Unjoined	map_keys
Pipelined/Unjoined	pipe_state
Pipelined/Unjoined	index_fact
Slowly Changing Dimensions	insert_dim
Slowly Changing Dimensions	index_dim
Transactional	map_keys
Transactional	load_trans
Transactional	ren_trans
Transactional	index_fact
Transactional/Inventory	map_keys

Transactional/Inventory	load_trans
Transactional/Inventory	inv_adjust
Transactional/Inventory	index_fact
Transactional/Inventory/ForceZero	map_keys
Transactional/Inventory/ForceZero	load_trans
Transactional/Inventory/ForceZero	force_zero
Transactional/Inventory/ForceZero	inv_adjust
Transactional/Inventory/ForceZero	index_fact
Transactional/Inventory/ForceZero/Unjoined	upd_unj
Transactional/Inventory/ForceZero/Unjoined	map_keys
Transactional/Inventory/ForceZero/Unjoined	load_trans
Transactional/Inventory/ForceZero/Unjoined	force_zero
Transactional/Inventory/ForceZero/Unjoined	inv_adjust
Transactional/Inventory/ForceZero/Unjoined	index_fact
Transactional/Inventory/Unjoined	upd_unj
Transactional/Inventory/Unjoined	map_keys
Transactional/Inventory/Unjoined	load_trans
Transactional/Inventory/Unjoined	inv_adjust
Transactional/Inventory/Unjoined	index_fact
Transactional/Statelike	map_keys
Transactional/Statelike	load_trans
Transactional/Statelike	load_state
Transactional/Statelike	index_fact
Transactional/Statelike/ForceClose	map_keys
Transactional/Statelike/ForceClose	load_trans
Transactional/Statelike/ForceClose	force_close
Transactional/Statelike/ForceClose	load_state
Transactional/Statelike/ForceClose	index_fact
Transactional/Statelike/ForceClose/Unjoined	upd_unj
Transactional/Statelike/ForceClose/Unjoined	map_keys
Transactional/Statelike/ForceClose/Unjoined	load_trans
Transactional/Statelike/ForceClose/Unjoined	force_close
Transactional/Statelike/ForceClose/Unjoined	load_state

Transactional/Statelike/ForceClose/Unjoined	index_fact
Transactional/Statelike/Unjoined	upd_unj
Transactional/Statelike/Unjoined	map_keys
Transactional/Statelike/Unjoined	load_trans
Transactional/Statelike/Unjoined	load_state
Transactional/Statelike/Unjoined	index_fact
Transactional/Unjoined	upd_unj
Transactional/Unjoined	map_keys
Transactional/Unjoined	load_trans
Transactional/Unjoined	ren_trans
Transactional/Unjoined	index_fact

The following are the pre-parsed pseudo-SQL source for the adaptive templates.

```
--#TEMPLATE_BEGIN# force_close
/*****
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- force_close
--
-- Close out deleted orders - those that no longer appear in the
-- staging table
--
-- SEE SAFETY VALVE BELOW
--
*****/

/*****
-- Delete temporary tables
*****/

--#BLOCK_BEGIN# DropTemps

$$DDL_BEGIN
$$DROP TABLE IF EXISTS[$$FCTTBL[]_FC]
$$DDL_END

--#BLOCK_END# DropTemps

/*****
-- Insert negative BOOKs for deleted orders
--
-- FC: ForceClose
*****/

--#BLOCK_BEGIN# MakeFC

$$SELECT INTO_BEGIN[$$FCTTBL[]_FC]
SELECT
    f.iss,
    f.ss_key,
    MAX(f.date_key) date_key,
    MIN(f.transtype key) transtype key,
```

```

MAX(f.seq) + 1 seq
, f.$$DIMKEYR_01
, f.$$DIMKEYR_02
, f.$$DIMKEYR_03
, f.$$DIMKEYR_04
, f.$$DIMKEYR_05
, f.$$DIMKEYR_06
, f.$$DIMKEYR_07
, f.$$DIMKEYR_08
, f.$$DIMKEYR_09
, f.$$DIMKEYR_10
, f.$$DEGKEY_01
, f.$$DEGKEY_02
, f.$$DEGKEY_03
,
-SUM(f.$$FCTCOL_001) $$FCTCOL_001
, -SUM(f.$$FCTCOL_002) $$FCTCOL_002
, -SUM(f.$$FCTCOL_003) $$FCTCOL_003
, -SUM(f.$$FCTCOL_004) $$FCTCOL_004
, -SUM(f.$$FCTCOL_005) $$FCTCOL_005
, -SUM(f.$$FCTCOL_006) $$FCTCOL_006
, -SUM(f.$$FCTCOL_007) $$FCTCOL_007
, -SUM(f.$$FCTCOL_008) $$FCTCOL_008
, -SUM(f.$$FCTCOL_009) $$FCTCOL_009
, -SUM(f.$$FCTCOL_010) $$FCTCOL_010
, -SUM(f.$$FCTCOL_011) $$FCTCOL_011
, -SUM(f.$$FCTCOL_012) $$FCTCOL_012
, -SUM(f.$$FCTCOL_013) $$FCTCOL_013
, -SUM(f.$$FCTCOL_014) $$FCTCOL_014
, -SUM(f.$$FCTCOL_015) $$FCTCOL_015
, -SUM(f.$$FCTCOL_016) $$FCTCOL_016
, -SUM(f.$$FCTCOL_017) $$FCTCOL_017
, -SUM(f.$$FCTCOL_018) $$FCTCOL_018
, -SUM(f.$$FCTCOL_019) $$FCTCOL_019
, -SUM(f.$$FCTCOL_020) $$FCTCOL_020
, -SUM(f.$$FCTCOL_021) $$FCTCOL_021
, -SUM(f.$$FCTCOL_022) $$FCTCOL_022
, -SUM(f.$$FCTCOL_023) $$FCTCOL_023
, -SUM(f.$$FCTCOL_024) $$FCTCOL_024
,
$$SELECT INTO BODY[$$FCTTBL[]_FC]
FROM
    $$FCTTBL[] $$CURR f
WHERE
    NOT EXISTS
        (SELECT 1 FROM $$FSTGTBL[]_MAP s WHERE s.iss = f.iss AND s.ss_key = f.ss_key)
GROUP BY
    f.iss,
    f.ss_key
, f.$$DIMKEYR_01
, f.$$DIMKEYR_02
, f.$$DIMKEYR_03
, f.$$DIMKEYR_04
, f.$$DIMKEYR_05
, f.$$DIMKEYR_06
, f.$$DIMKEYR_07
, f.$$DIMKEYR_08
, f.$$DIMKEYR_09
, f.$$DIMKEYR_10
, f.$$DEGKEY_01
, f.$$DEGKEY_02
, f.$$DEGKEY_03
,
HAVING
    (
        (SUM(f.$$FCTCOL_001) <> 0)
    OR
        (SUM(f.$$FCTCOL_002) <> 0)
    OR
        (SUM(f.$$FCTCOL_003) <> 0)
    OR
        (SUM(f.$$FCTCOL_004) <> 0)
    OR
        (SUM(f.$$FCTCOL_005) <> 0)
    OR
        (SUM(f.$$FCTCOL_006) <> 0)
    )

```

```

OR      (SUM(f.$$FCTCOL_007) <> 0)
OR      (SUM(f.$$FCTCOL_008) <> 0)
OR      (SUM(f.$$FCTCOL_009) <> 0)
OR      (SUM(f.$$FCTCOL_010) <> 0)
OR      (SUM(f.$$FCTCOL_011) <> 0)
OR      (SUM(f.$$FCTCOL_012) <> 0)
OR      (SUM(f.$$FCTCOL_013) <> 0)
OR      (SUM(f.$$FCTCOL_014) <> 0)
OR      (SUM(f.$$FCTCOL_015) <> 0)
OR      (SUM(f.$$FCTCOL_016) <> 0)
OR      (SUM(f.$$FCTCOL_017) <> 0)
OR      (SUM(f.$$FCTCOL_018) <> 0)
OR      (SUM(f.$$FCTCOL_019) <> 0)
OR      (SUM(f.$$FCTCOL_020) <> 0)
OR      (SUM(f.$$FCTCOL_021) <> 0)
OR      (SUM(f.$$FCTCOL_022) <> 0)
OR      (SUM(f.$$FCTCOL_023) <> 0)
OR      (SUM(f.$$FCTCOL_024) <> 0)
)

```

```

AND      MIN(f.transtype_key) <= 99

```

```

AND      MIN(f.transtype_key) >= 1

```

```

--#BLOCK_END# MakeFC

```

```

/*****
-- SAFETY VALVE - THIS PROC ONLY DOES ANYTHING
-- IF THE STAGING TABLE HAS AT LEAST ONE ROW
*****/

```

```

--#BLOCK_BEGIN# SafetyValue

```

```

DECLARE $$VAR[count_MAP] $$EPIINT$$EOS

```

```

BEGIN

```

```

$$VAR_ASSIGN_BEGIN[count_MAP]
SELECT COUNT(1)
$$VAR_ASSIGN INTO[count_MAP]
FROM $$FSTGTBL[]_MAP
$$VAR_ASSIGN_END

```

```

$$IF[($$VAR[count_MAP] = 0)]
DELETE FROM $$FCTTBL[]_FC$$EOS
$$END_IF

```

```

END$$EOS

```

```

--#BLOCK_END# SafetyValue

```

```

/*****
-- Count processed, inserted rows
*****/

```

```

--#BLOCK_BEGIN# SPResults

```

```

BEGIN

```

```

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM $$FCTTBL[]_FC$$EOS

```

```

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', COUNT(1) FROM $$FCTTBL[]_FC$$EOS

```

```

END$$EOS

```

```

--#BLOCK_END# SPResults

```

```

--#TEMPLATE_END# force_close

```

```

--#TEMPLATE_BEGIN# load_state

/*****
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- load_state
--
-- Load order bookings into fact table by creating transactional
-- data from state data
--
-- load_trans must be run before this procedure to create TIN table
--
*****/

/*****
-- Delete temporary tables
*****/

--#BLOCK_BEGIN# DropTemps

$$DDL_BEGIN
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_MFL]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_1ST]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IL]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IR]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IRD]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IND]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_NFD]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IRM]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IDM]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_ILM]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IMI]
$$DDL_END

--#BLOCK_END# DropTemps

/*****
-- Set join order for SQL Server
*****/

--#BLOCK_BEGIN# ForcePlanOn

$$SQLSERVER[SET FORCEPLAN ON]

--#BLOCK_END# ForcePlanOn

/*****
-- Remove rows older than fact table - history can not be rewritten - only
-- the last date for an order can be changed. Note that we compare transtype's
-- because SHIP type transactions might occur at a later date and we don't want
-- those to interfere
--
-- Also, since the staging table may have multiple entries for a given order on
-- a single day - we assume that the list one inserted in the Staging table will
-- be used (since ikey is an IDENTITY column)
--
-- Note that a given ss_key must use the same Booking transtype for all of time,
-- otherwise the transtype_key
--
-- MFL: Mapped Filtered
*****/

--#BLOCK_BEGIN# MakeMFL

$$SELECT_INTO_BEGIN[$$FCTTBL[]_MFL]
SELECT
    s.*
$$SELECT_INTO_BODY[$$FCTTBL[]_MFL]
FROM

```

```

WHERE    $$FSTGTBL[]_MAP s, bus_process b
        ((s.date_key >= (SELECT MAX(date_key) FROM $$FCTTBL[]$$CURR f WHERE
            s.iss = f.iss AND s.ss_key = f.ss_key AND
            s.transtype_key = f.transtype_key))
        OR NOT EXISTS (SELECT * FROM $$FCTTBL[]$$CURR f WHERE
            s.iss = f.iss AND s.ss_key = f.ss_key AND
            s.transtype_key = f.transtype_key))
AND      s.ikey = (SELECT MAX(t.ikey) FROM $$FSTGTBL[]_MAP t WHERE
            s.iss = t.iss AND
            s.ss_key = t.ss_key AND
            s.date_key = t.date_key AND
            t.process_key = b.process_key)
AND
        s.process_key = b.process_key AND b.process_name = 'LoadState'

--#BLOCK_END# MakeMFL

/*****
-- Index MFL table for later queries
*****/

--#BLOCK_BEGIN# IndexMFL

$$DDL_BEGIN
$$DDL_EXEC[
CREATE INDEX X$$FCTTBL[]_MFL ON $$FCTTBL[]_MFL
(
    iss, ss_key, date_key
)
]
$$DDL_END

--#BLOCK_END# IndexMFL

/*****
-- Get oldest state rows for each unique sskey
--
-- We need to treat the first entry for each order
-- in the staging table separately from all others, since
-- only the first entry needs to be compared with
-- already existing fact entry rows to create transactions.
-- All subsequent dates for that order in the Fact table
-- can be delta'd with other staging table entries - see the
-- section below on Pairwise deltas.
--
-- MFL should be indexed
--
-- 1ST: The first record for each iss, ss_key
*****/

--#BLOCK_BEGIN# Make1ST

$$SELECT_INTO_BEGIN[$$FCTTBL[]_1ST]
SELECT
    s.*
$$SELECT_INTO_BODY[$$FCTTBL[]_1ST]
FROM
    $$FCTTBL[]_MFL s
WHERE
    s.date_key = (SELECT MIN(date_key) FROM $$FCTTBL[]_MFL t WHERE
        s.iss = t.iss AND s.ss_key = t.ss_key)

--#BLOCK_END# Make1ST

/*****
-- Index 1ST for later queries
*****/

--#BLOCK_BEGIN# Index1ST

```

```

$$DDL_BEGIN
$$DDL_EXEC[
CREATE UNIQUE INDEX XPK$$FCTTBL[]_1ST ON $$FCTTBL[]_1ST
(
    iss, ss_key
)
]
$$DDL_END

--#BLOCK_END# Index1ST

/*****
-- Insert negative BOOKs for changed dim keys
--
-- This query will add up all existing Books and Loss's
-- for this order and the net facts will be cancelled out
-- with the old Dimension keys. Note that an invariant of this
-- procedure is that only one set of dimensions at a time
-- can have non-zero facts.
--
-- Fact table Should be indexed
--
-- HAVING Clause is needed to prevent changing of dimensions
-- on fully shipped order from causing a transaction - no sense
-- creating fact rows with all zero's in them
--
-- Note that we increment the sequence number just in case
-- this new transaction occurs on the same date as the last
-- existing one in the fact table - to avoid index errors
--
-- IL: InsertLost
*****/

--#BLOCK_BEGIN# MakeIL

$$SELECT_INTO_BEGIN[$$FCTTBL[]_IL]
SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    MAX(f.seq) + 1 seq
    ,
    f.$$DIMKEYR_01
    ,
    f.$$DIMKEYR_02
    ,
    f.$$DIMKEYR_03
    ,
    f.$$DIMKEYR_04
    ,
    f.$$DIMKEYR_05
    ,
    f.$$DIMKEYR_06
    ,
    f.$$DIMKEYR_07
    ,
    f.$$DIMKEYR_08
    ,
    f.$$DIMKEYR_09
    ,
    f.$$DIMKEYR_10
    ,
    f.$$DEGKEY_01
    ,
    f.$$DEGKEY_02
    ,
    f.$$DEGKEY_03
    ,
    -SUM(f.$$FCTCOL_001) $$FCTCOL_001
    ,
    -SUM(f.$$FCTCOL_002) $$FCTCOL_002
    ,
    -SUM(f.$$FCTCOL_003) $$FCTCOL_003
    ,
    -SUM(f.$$FCTCOL_004) $$FCTCOL_004
    ,
    -SUM(f.$$FCTCOL_005) $$FCTCOL_005
    ,
    -SUM(f.$$FCTCOL_006) $$FCTCOL_006
    ,
    -SUM(f.$$FCTCOL_007) $$FCTCOL_007
    ,
    -SUM(f.$$FCTCOL_008) $$FCTCOL_008
    ,
    -SUM(f.$$FCTCOL_009) $$FCTCOL_009
    ,
    -SUM(f.$$FCTCOL_010) $$FCTCOL_010
    ,
    -SUM(f.$$FCTCOL_011) $$FCTCOL_011
    ,
    -SUM(f.$$FCTCOL_012) $$FCTCOL_012
    ,
    -SUM(f.$$FCTCOL_013) $$FCTCOL_013
    ,
    -SUM(f.$$FCTCOL_014) $$FCTCOL_014
    ,
    -SUM(f.$$FCTCOL_015) $$FCTCOL_015

```



```

, -SUM(f.$$FCTCOL_016) $$FCTCOL_016
, -SUM(f.$$FCTCOL_017) $$FCTCOL_017
, -SUM(f.$$FCTCOL_018) $$FCTCOL_018
, -SUM(f.$$FCTCOL_019) $$FCTCOL_019
, -SUM(f.$$FCTCOL_020) $$FCTCOL_020
, -SUM(f.$$FCTCOL_021) $$FCTCOL_021
, -SUM(f.$$FCTCOL_022) $$FCTCOL_022
, -SUM(f.$$FCTCOL_023) $$FCTCOL_023
, -SUM(f.$$FCTCOL_024) $$FCTCOL_024

$$SELECT INTO_BODY[$$FCTTBL[]_IL]
FROM
    $$FCTTBL[]_1ST s, $$FCTTBL[]$$CURR f
WHERE
    s.iss = f.iss AND s.ss_key = f.ss_key
AND
    ((s.$$DIMKEYR_06 <> f.$$DIMKEYR_06) OR
    (s.$$DIMKEYR_05 <> f.$$DIMKEYR_05) OR
    (s.$$DIMKEYR_07 <> f.$$DIMKEYR_07) OR
    (s.$$DIMKEYR_04 <> f.$$DIMKEYR_04) OR
    (s.$$DIMKEYR_08 <> f.$$DIMKEYR_08) OR
    (s.$$DIMKEYR_03 <> f.$$DIMKEYR_03) OR
    (s.$$DIMKEYR_09 <> f.$$DIMKEYR_09) OR
    (s.$$DIMKEYR_02 <> f.$$DIMKEYR_02) OR
    (s.$$DIMKEYR_10 <> f.$$DIMKEYR_10) OR
    (s.$$DIMKEYR_01 <> f.$$DIMKEYR_01) )
GROUP BY
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key
, f.$$DIMKEYR_01
, f.$$DIMKEYR_02
, f.$$DIMKEYR_03
, f.$$DIMKEYR_04
, f.$$DIMKEYR_05
, f.$$DIMKEYR_06
, f.$$DIMKEYR_07
, f.$$DIMKEYR_08
, f.$$DIMKEYR_09
, f.$$DIMKEYR_10
, f.$$DEGKEY_01
, f.$$DEGKEY_02
, f.$$DEGKEY_03

HAVING
    MIN(f.transtype_key) = s.transtype_key
AND
    (
    (SUM(f.$$FCTCOL_001) <> 0)
OR
    (SUM(f.$$FCTCOL_002) <> 0)
OR
    (SUM(f.$$FCTCOL_003) <> 0)
OR
    (SUM(f.$$FCTCOL_004) <> 0)
OR
    (SUM(f.$$FCTCOL_005) <> 0)
OR
    (SUM(f.$$FCTCOL_006) <> 0)
OR
    (SUM(f.$$FCTCOL_007) <> 0)
OR
    (SUM(f.$$FCTCOL_008) <> 0)
OR
    (SUM(f.$$FCTCOL_009) <> 0)
OR
    (SUM(f.$$FCTCOL_010) <> 0)
OR
    (SUM(f.$$FCTCOL_011) <> 0)
OR
    (SUM(f.$$FCTCOL_012) <> 0)
OR
    (SUM(f.$$FCTCOL_013) <> 0)
OR
    (SUM(f.$$FCTCOL_014) <> 0)
OR
    (SUM(f.$$FCTCOL_015) <> 0)
OR
    (SUM(f.$$FCTCOL_016) <> 0)
OR
    (SUM(f.$$FCTCOL_017) <> 0)
OR
    (SUM(f.$$FCTCOL_018) <> 0)
OR
    (SUM(f.$$FCTCOL_019) <> 0)
OR
    (SUM(f.$$FCTCOL_020) <> 0)
OR
    (SUM(f.$$FCTCOL_021) <> 0)
OR
    (SUM(f.$$FCTCOL_022) <> 0)
    )

```

```

OR      (SUM(f.$$FCTCOL_023) <> 0)
OR      (SUM(f.$$FCTCOL_024) <> 0)
)

--#BLOCK_END# MakeIL

/*****
-- Index IL for later queries
*****/

--#BLOCK_BEGIN# IndexIL

$$DDL_BEGIN
$$DDL EXEC[
CREATE INDEX XPK$$FCTTBL[]_IL ON $$FCTTBL[]_IL
(
    iss, ss_key
)
]
$$DDL_END

--#BLOCK_END# IndexIL

/*****
-- Insert BOOKs for changed dim keys
--
-- When a dimension changes then just create a booking
-- transaction for whatever we negated above with the new
-- dimension and fact values
--
-- 1ST should be indexed
--
-- Note that we add one to whatever we used as the last
-- seq because this transaction occurs on the same
-- date as the negative one above
--
-- IR: Insert Rebook
*****/

--#BLOCK_BEGIN# MakeIR

$$SELECT_INTO_BEGIN[$$FCTTBL[]_IR]
SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    l.transtype_key,
    l.seq + 1 seq
    , s.$$DIMKEYR_01
    , s.$$DIMKEYR_02
    , s.$$DIMKEYR_03
    , s.$$DIMKEYR_04
    , s.$$DIMKEYR_05
    , s.$$DIMKEYR_06
    , s.$$DIMKEYR_07
    , s.$$DIMKEYR_08
    , s.$$DIMKEYR_09
    , s.$$DIMKEYR_10
    , s.$$DEGKEY_01
    , s.$$DEGKEY_02
    , s.$$DEGKEY_03

    , -1.$$FCTCOL_001 $$FCTCOL_001
    , -1.$$FCTCOL_002 $$FCTCOL_002
    , -1.$$FCTCOL_003 $$FCTCOL_003
    , -1.$$FCTCOL_004 $$FCTCOL_004
    , -1.$$FCTCOL_005 $$FCTCOL_005
    , -1.$$FCTCOL_006 $$FCTCOL_006
    , -1.$$FCTCOL_007 $$FCTCOL_007
    , -1.$$FCTCOL_008 $$FCTCOL_008
    , -1.$$FCTCOL_009 $$FCTCOL_009

```

```

-1.$$FCTCOL_010 $$FCTCOL_010
-1.$$FCTCOL_011 $$FCTCOL_011
-1.$$FCTCOL_012 $$FCTCOL_012
-1.$$FCTCOL_013 $$FCTCOL_013
-1.$$FCTCOL_014 $$FCTCOL_014
-1.$$FCTCOL_015 $$FCTCOL_015
-1.$$FCTCOL_016 $$FCTCOL_016
-1.$$FCTCOL_017 $$FCTCOL_017
-1.$$FCTCOL_018 $$FCTCOL_018
-1.$$FCTCOL_019 $$FCTCOL_019
-1.$$FCTCOL_020 $$FCTCOL_020
-1.$$FCTCOL_021 $$FCTCOL_021
-1.$$FCTCOL_022 $$FCTCOL_022
-1.$$FCTCOL_023 $$FCTCOL_023
-1.$$FCTCOL_024 $$FCTCOL_024

$$SELECT INTO BODY[$$FCTBL[]_IR]
FROM
    $$FCTBL[]_IL 1, $$FCTBL[]_1ST s
WHERE 1.iss = s.iss AND 1.ss_key = s.ss_key

--#BLOCK_END# MakeIR

/*****
-- Insert BOOKs for changed dim keys where fact
-- also changed
--
-- When a dimension changes at the same time as
-- a fact then we need to make up the fact difference
--
-- 1ST should be indexed
--
-- Note that we add two to whatever we used as the last
-- seq because this transaction occurs on the same
-- date as the negative and positive ones above
--
-- Note also that the Left Outer join uses transtype_key
-- so that only the Bookings at the old value will be counted.
-- Whereas above for the negative transaction value
-- we want to include Shipments in our calculation, here
-- we only want to see how Booking Facts have changed.
--
-- Here again, only one Booking transaction type is supported
-- per ss_key
--
-- IRD: Insert Rebook delta
*****/

--#BLOCK_BEGIN# MakeIRD

$$SELECT INTO BEGIN[$$FCTBL[]_IRD]
SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    1.seq + 2 seq
,
    s.$$DIMKEYR_01
,
    s.$$DIMKEYR_02
,
    s.$$DIMKEYR_03
,
    s.$$DIMKEYR_04
,
    s.$$DIMKEYR_05
,
    s.$$DIMKEYR_06
,
    s.$$DIMKEYR_07
,
    s.$$DIMKEYR_08
,
    s.$$DIMKEYR_09
,
    s.$$DIMKEYR_10
,
    s.$$DEGKEY_01
,
    s.$$DEGKEY_02
,
    s.$$DEGKEY_03

```

```

MAX(s.$FCTCOL_001)-$NVL(SUM(f.$FCTCOL_001)~,~0)] $FCTCOL_001
MAX(s.$FCTCOL_002)-$NVL(SUM(f.$FCTCOL_002)~,~0)] $FCTCOL_002
MAX(s.$FCTCOL_003)-$NVL(SUM(f.$FCTCOL_003)~,~0)] $FCTCOL_003
MAX(s.$FCTCOL_004)-$NVL(SUM(f.$FCTCOL_004)~,~0)] $FCTCOL_004
MAX(s.$FCTCOL_005)-$NVL(SUM(f.$FCTCOL_005)~,~0)] $FCTCOL_005
MAX(s.$FCTCOL_006)-$NVL(SUM(f.$FCTCOL_006)~,~0)] $FCTCOL_006
MAX(s.$FCTCOL_007)-$NVL(SUM(f.$FCTCOL_007)~,~0)] $FCTCOL_007
MAX(s.$FCTCOL_008)-$NVL(SUM(f.$FCTCOL_008)~,~0)] $FCTCOL_008
MAX(s.$FCTCOL_009)-$NVL(SUM(f.$FCTCOL_009)~,~0)] $FCTCOL_009
MAX(s.$FCTCOL_010)-$NVL(SUM(f.$FCTCOL_010)~,~0)] $FCTCOL_010
MAX(s.$FCTCOL_011)-$NVL(SUM(f.$FCTCOL_011)~,~0)] $FCTCOL_011
MAX(s.$FCTCOL_012)-$NVL(SUM(f.$FCTCOL_012)~,~0)] $FCTCOL_012
MAX(s.$FCTCOL_013)-$NVL(SUM(f.$FCTCOL_013)~,~0)] $FCTCOL_013
MAX(s.$FCTCOL_014)-$NVL(SUM(f.$FCTCOL_014)~,~0)] $FCTCOL_014
MAX(s.$FCTCOL_015)-$NVL(SUM(f.$FCTCOL_015)~,~0)] $FCTCOL_015
MAX(s.$FCTCOL_016)-$NVL(SUM(f.$FCTCOL_016)~,~0)] $FCTCOL_016
MAX(s.$FCTCOL_017)-$NVL(SUM(f.$FCTCOL_017)~,~0)] $FCTCOL_017
MAX(s.$FCTCOL_018)-$NVL(SUM(f.$FCTCOL_018)~,~0)] $FCTCOL_018
MAX(s.$FCTCOL_019)-$NVL(SUM(f.$FCTCOL_019)~,~0)] $FCTCOL_019
MAX(s.$FCTCOL_020)-$NVL(SUM(f.$FCTCOL_020)~,~0)] $FCTCOL_020
MAX(s.$FCTCOL_021)-$NVL(SUM(f.$FCTCOL_021)~,~0)] $FCTCOL_021
MAX(s.$FCTCOL_022)-$NVL(SUM(f.$FCTCOL_022)~,~0)] $FCTCOL_022
MAX(s.$FCTCOL_023)-$NVL(SUM(f.$FCTCOL_023)~,~0)] $FCTCOL_023
MAX(s.$FCTCOL_024)-$NVL(SUM(f.$FCTCOL_024)~,~0)] $FCTCOL_024

$$SELECT INTO BODY[$FCTBL[]_IRD]
FROM
    $$FCTBL[]_IL 1, $$FCTBL[]_1ST s
    $$LOJ FROM[$FCTBL[]_$$CURR f ~,~ s.iss = f.iss AND s.ss_key = f.ss_key AND
s.transtype_key = f.transtype_key]
WHERE
    l.iss = s.iss AND l.ss_key = s.ss_key
$$JOIN WHERE[s.iss = f.iss (+) AND s.ss_key = f.ss_key (+) AND s.transtype_key =
f.transtype_key (+)]
GROUP BY
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    l.seq
    s.$DIMKEYR_01
    s.$DIMKEYR_02
    s.$DIMKEYR_03
    s.$DIMKEYR_04
    s.$DIMKEYR_05
    s.$DIMKEYR_06
    s.$DIMKEYR_07
    s.$DIMKEYR_08
    s.$DIMKEYR_09
    s.$DIMKEYR_10
    s.$DEGKEY_01
    s.$DEGKEY_02
    s.$DEGKEY_03

HAVING
    ($$NVL(SUM(f.$FCTCOL_001)~,~0)] <> MAX(s.$FCTCOL_001))
OR
    ($$NVL(SUM(f.$FCTCOL_002)~,~0)] <> MAX(s.$FCTCOL_002))
OR
    ($$NVL(SUM(f.$FCTCOL_003)~,~0)] <> MAX(s.$FCTCOL_003))
OR
    ($$NVL(SUM(f.$FCTCOL_004)~,~0)] <> MAX(s.$FCTCOL_004))
OR
    ($$NVL(SUM(f.$FCTCOL_005)~,~0)] <> MAX(s.$FCTCOL_005))
OR
    ($$NVL(SUM(f.$FCTCOL_006)~,~0)] <> MAX(s.$FCTCOL_006))
OR
    ($$NVL(SUM(f.$FCTCOL_007)~,~0)] <> MAX(s.$FCTCOL_007))
OR
    ($$NVL(SUM(f.$FCTCOL_008)~,~0)] <> MAX(s.$FCTCOL_008))
OR
    ($$NVL(SUM(f.$FCTCOL_009)~,~0)] <> MAX(s.$FCTCOL_009))
OR
    ($$NVL(SUM(f.$FCTCOL_010)~,~0)] <> MAX(s.$FCTCOL_010))
OR
    ($$NVL(SUM(f.$FCTCOL_011)~,~0)] <> MAX(s.$FCTCOL_011))
OR
    ($$NVL(SUM(f.$FCTCOL_012)~,~0)] <> MAX(s.$FCTCOL_012))
OR
    ($$NVL(SUM(f.$FCTCOL_013)~,~0)] <> MAX(s.$FCTCOL_013))
OR
    ($$NVL(SUM(f.$FCTCOL_014)~,~0)] <> MAX(s.$FCTCOL_014))
OR
    ($$NVL(SUM(f.$FCTCOL_015)~,~0)] <> MAX(s.$FCTCOL_015))
OR
    ($$NVL(SUM(f.$FCTCOL_016)~,~0)] <> MAX(s.$FCTCOL_016))

```

```

OR      ($$NVL[SUM(f.$$FCTCOL_017) ~,~ 0] <> MAX(s.$$FCTCOL_017))
OR      ($$NVL[SUM(f.$$FCTCOL_018) ~,~ 0] <> MAX(s.$$FCTCOL_018))
OR      ($$NVL[SUM(f.$$FCTCOL_019) ~,~ 0] <> MAX(s.$$FCTCOL_019))
OR      ($$NVL[SUM(f.$$FCTCOL_020) ~,~ 0] <> MAX(s.$$FCTCOL_020))
OR      ($$NVL[SUM(f.$$FCTCOL_021) ~,~ 0] <> MAX(s.$$FCTCOL_021))
OR      ($$NVL[SUM(f.$$FCTCOL_022) ~,~ 0] <> MAX(s.$$FCTCOL_022))
OR      ($$NVL[SUM(f.$$FCTCOL_023) ~,~ 0] <> MAX(s.$$FCTCOL_023))
OR      ($$NVL[SUM(f.$$FCTCOL_024) ~,~ 0] <> MAX(s.$$FCTCOL_024))

--#BLOCK_END# MakeIRD

/*****
-- Insert BOOKs for deltas with same dim keys OR for
-- brand new orders.
--
-- Note that we DON'T want to count Shipments
-- (so shipment ss_key's should be different from
-- order ss_keys) since we just want bookings to sum up
-- to whatever this transaction says they should be.
--
-- Fact table should be indexed
--
-- WHERE clause prevents double booking on changed
-- dimension - if we didn't use the NOT EXISTS clause
-- then this query would repeat the work of the last one
-- above - which we have already taken care of
--
-- HAVING clause ensures that multiple 0 records don't
-- get inserted whenever this procedure is run
--
-- Note that we increment the sequence number just in case
-- this new transaction occurs on the same date as the last
-- existing one in the fact table - to avoid index errors
--
-- IND: Insert New Delta
*****/

--#BLOCK_BEGIN# MakeIND

$$SELECT INTO BEGIN[$$FCTTBL[]_IND]
SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    $$NVL[MAX(f.seq) ~,~ 0] + 1 seq
    ,
    s.$$DIMKEYR_01
    ,
    s.$$DIMKEYR_02
    ,
    s.$$DIMKEYR_03
    ,
    s.$$DIMKEYR_04
    ,
    s.$$DIMKEYR_05
    ,
    s.$$DIMKEYR_06
    ,
    s.$$DIMKEYR_07
    ,
    s.$$DIMKEYR_08
    ,
    s.$$DIMKEYR_09
    ,
    s.$$DIMKEYR_10
    ,
    s.$$DEGKEY_01
    ,
    s.$$DEGKEY_02
    ,
    s.$$DEGKEY_03
    ,
    MAX(s.$$FCTCOL_001)-$$NVL[SUM(f.$$FCTCOL_001) ~,~ 0] $$FCTCOL_001
    ,
    MAX(s.$$FCTCOL_002)-$$NVL[SUM(f.$$FCTCOL_002) ~,~ 0] $$FCTCOL_002
    ,
    MAX(s.$$FCTCOL_003)-$$NVL[SUM(f.$$FCTCOL_003) ~,~ 0] $$FCTCOL_003
    ,
    MAX(s.$$FCTCOL_004)-$$NVL[SUM(f.$$FCTCOL_004) ~,~ 0] $$FCTCOL_004
    ,
    MAX(s.$$FCTCOL_005)-$$NVL[SUM(f.$$FCTCOL_005) ~,~ 0] $$FCTCOL_005
    ,
    MAX(s.$$FCTCOL_006)-$$NVL[SUM(f.$$FCTCOL_006) ~,~ 0] $$FCTCOL_006
    ,
    MAX(s.$$FCTCOL_007)-$$NVL[SUM(f.$$FCTCOL_007) ~,~ 0] $$FCTCOL_007
    ,
    MAX(s.$$FCTCOL_008)-$$NVL[SUM(f.$$FCTCOL_008) ~,~ 0] $$FCTCOL_008
    ,
    MAX(s.$$FCTCOL_009)-$$NVL[SUM(f.$$FCTCOL_009) ~,~ 0] $$FCTCOL_009
    ,
    MAX(s.$$FCTCOL_010)-$$NVL[SUM(f.$$FCTCOL_010) ~,~ 0] $$FCTCOL_010
    ,
    MAX(s.$$FCTCOL_011)-$$NVL[SUM(f.$$FCTCOL_011) ~,~ 0] $$FCTCOL_011

```

```

MAX(s.$FCTCOL_012)-$$NVL(SUM(f.$FCTCOL_012)~,~0] $$FCTCOL_012
MAX(s.$FCTCOL_013)-$$NVL(SUM(f.$FCTCOL_013)~,~0] $$FCTCOL_013
MAX(s.$FCTCOL_014)-$$NVL(SUM(f.$FCTCOL_014)~,~0] $$FCTCOL_014
MAX(s.$FCTCOL_015)-$$NVL(SUM(f.$FCTCOL_015)~,~0] $$FCTCOL_015
MAX(s.$FCTCOL_016)-$$NVL(SUM(f.$FCTCOL_016)~,~0] $$FCTCOL_016
MAX(s.$FCTCOL_017)-$$NVL(SUM(f.$FCTCOL_017)~,~0] $$FCTCOL_017
MAX(s.$FCTCOL_018)-$$NVL(SUM(f.$FCTCOL_018)~,~0] $$FCTCOL_018
MAX(s.$FCTCOL_019)-$$NVL(SUM(f.$FCTCOL_019)~,~0] $$FCTCOL_019
MAX(s.$FCTCOL_020)-$$NVL(SUM(f.$FCTCOL_020)~,~0] $$FCTCOL_020
MAX(s.$FCTCOL_021)-$$NVL(SUM(f.$FCTCOL_021)~,~0] $$FCTCOL_021
MAX(s.$FCTCOL_022)-$$NVL(SUM(f.$FCTCOL_022)~,~0] $$FCTCOL_022
MAX(s.$FCTCOL_023)-$$NVL(SUM(f.$FCTCOL_023)~,~0] $$FCTCOL_023
MAX(s.$FCTCOL_024)-$$NVL(SUM(f.$FCTCOL_024)~,~0] $$FCTCOL_024

$$SELECT INTO BODY[$$FCTTBL[]_IND]
FROM
    $$FCTTBL[]_1ST s $$LOJ FROM[$$FCTTBL[]]$CURR f ~,~
    s.iss = f.iss AND s.ss_key = f.ss_key AND f.transtype_key = s.transtype_key]
WHERE
    NOT EXISTS (SELECT * FROM $$FCTTBL[]_IL WHERE iss = s.iss AND ss_key = s.ss_key)
$$JOIN WHERE[s.iss = f.iss (+) AND s.ss_key = f.ss_key (+) AND s.transtype_key =
f.transtype_key (+)]
GROUP BY
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key
    s.$DIMKEYR_01
    s.$DIMKEYR_02
    s.$DIMKEYR_03
    s.$DIMKEYR_04
    s.$DIMKEYR_05
    s.$DIMKEYR_06
    s.$DIMKEYR_07
    s.$DIMKEYR_08
    s.$DIMKEYR_09
    s.$DIMKEYR_10
    s.$DEGKEY_01
    s.$DEGKEY_02
    s.$DEGKEY_03

HAVING
    ($$NVL(SUM(f.$FCTCOL_001)~,~0] <> MAX(s.$FCTCOL_001))
OR
    ($$NVL(SUM(f.$FCTCOL_002)~,~0] <> MAX(s.$FCTCOL_002))
OR
    ($$NVL(SUM(f.$FCTCOL_003)~,~0] <> MAX(s.$FCTCOL_003))
OR
    ($$NVL(SUM(f.$FCTCOL_004)~,~0] <> MAX(s.$FCTCOL_004))
OR
    ($$NVL(SUM(f.$FCTCOL_005)~,~0] <> MAX(s.$FCTCOL_005))
OR
    ($$NVL(SUM(f.$FCTCOL_006)~,~0] <> MAX(s.$FCTCOL_006))
OR
    ($$NVL(SUM(f.$FCTCOL_007)~,~0] <> MAX(s.$FCTCOL_007))
OR
    ($$NVL(SUM(f.$FCTCOL_008)~,~0] <> MAX(s.$FCTCOL_008))
OR
    ($$NVL(SUM(f.$FCTCOL_009)~,~0] <> MAX(s.$FCTCOL_009))
OR
    ($$NVL(SUM(f.$FCTCOL_010)~,~0] <> MAX(s.$FCTCOL_010))
OR
    ($$NVL(SUM(f.$FCTCOL_011)~,~0] <> MAX(s.$FCTCOL_011))
OR
    ($$NVL(SUM(f.$FCTCOL_012)~,~0] <> MAX(s.$FCTCOL_012))
OR
    ($$NVL(SUM(f.$FCTCOL_013)~,~0] <> MAX(s.$FCTCOL_013))
OR
    ($$NVL(SUM(f.$FCTCOL_014)~,~0] <> MAX(s.$FCTCOL_014))
OR
    ($$NVL(SUM(f.$FCTCOL_015)~,~0] <> MAX(s.$FCTCOL_015))
OR
    ($$NVL(SUM(f.$FCTCOL_016)~,~0] <> MAX(s.$FCTCOL_016))
OR
    ($$NVL(SUM(f.$FCTCOL_017)~,~0] <> MAX(s.$FCTCOL_017))
OR
    ($$NVL(SUM(f.$FCTCOL_018)~,~0] <> MAX(s.$FCTCOL_018))
OR
    ($$NVL(SUM(f.$FCTCOL_019)~,~0] <> MAX(s.$FCTCOL_019))
OR
    ($$NVL(SUM(f.$FCTCOL_020)~,~0] <> MAX(s.$FCTCOL_020))
OR
    ($$NVL(SUM(f.$FCTCOL_021)~,~0] <> MAX(s.$FCTCOL_021))
OR
    ($$NVL(SUM(f.$FCTCOL_022)~,~0] <> MAX(s.$FCTCOL_022))
OR
    ($$NVL(SUM(f.$FCTCOL_023)~,~0] <> MAX(s.$FCTCOL_023))
OR
    ($$NVL(SUM(f.$FCTCOL_024)~,~0] <> MAX(s.$FCTCOL_024))

--#BLOCK_END# MakeIND

/*****

```

```

-- Form pairwise deltas for all rows except earliest for each sskey
--
-- Each row created in NFD will consist of two sequential entries from the
-- staing table. So if N enties for an order exist in MFL (after we have filtered
-- out same-date duplicates) then all the queries above will deal with the earliest entry,
whereas
-- all the queries below (including this one) will deal with the N-1 deltaing transactions
--
-- This query assumes that MFL will already have been filtered
-- to have a single record for each order/datekey
--
-- NFD: Not First Delta
/*****
--#BLOCK_BEGIN# MakeNFD

$$SELECT INTO_BEGIN[$$FCTTBL[]_NFD]
SELECT
    s.iss siss, t.iss tiss
    , s.ss_key sss_key, t.ss_key tss_key
    , s.date_key sdate_key, t.date_key tdate_key
    , s.transtype_key stranstype_key, t.transtype_key ttranstype_key
    , s.$$DIMKEYR_01 s$$DIMKEYR_01, t.$$DIMKEYR_01 t$$DIMKEYR_01
    , s.$$DIMKEYR_02 s$$DIMKEYR_02, t.$$DIMKEYR_02 t$$DIMKEYR_02
    , s.$$DIMKEYR_03 s$$DIMKEYR_03, t.$$DIMKEYR_03 t$$DIMKEYR_03
    , s.$$DIMKEYR_04 s$$DIMKEYR_04, t.$$DIMKEYR_04 t$$DIMKEYR_04
    , s.$$DIMKEYR_05 s$$DIMKEYR_05, t.$$DIMKEYR_05 t$$DIMKEYR_05
    , s.$$DIMKEYR_06 s$$DIMKEYR_06, t.$$DIMKEYR_06 t$$DIMKEYR_06
    , s.$$DIMKEYR_07 s$$DIMKEYR_07, t.$$DIMKEYR_07 t$$DIMKEYR_07
    , s.$$DIMKEYR_08 s$$DIMKEYR_08, t.$$DIMKEYR_08 t$$DIMKEYR_08
    , s.$$DIMKEYR_09 s$$DIMKEYR_09, t.$$DIMKEYR_09 t$$DIMKEYR_09
    , s.$$DIMKEYR_10 s$$DIMKEYR_10, t.$$DIMKEYR_10 t$$DIMKEYR_10
    , s.$$DEGKEY_01 s$$DEGKEY_01, t.$$DEGKEY_01 t$$DEGKEY_01
    , s.$$DEGKEY_02 s$$DEGKEY_02, t.$$DEGKEY_02 t$$DEGKEY_02
    , s.$$DEGKEY_03 s$$DEGKEY_03, t.$$DEGKEY_03 t$$DEGKEY_03
    , s.$$FCTCOL_001 s$$FCTCOL_001, t.$$FCTCOL_001 t$$FCTCOL_001
    , s.$$FCTCOL_002 s$$FCTCOL_002, t.$$FCTCOL_002 t$$FCTCOL_002
    , s.$$FCTCOL_003 s$$FCTCOL_003, t.$$FCTCOL_003 t$$FCTCOL_003
    , s.$$FCTCOL_004 s$$FCTCOL_004, t.$$FCTCOL_004 t$$FCTCOL_004
    , s.$$FCTCOL_005 s$$FCTCOL_005, t.$$FCTCOL_005 t$$FCTCOL_005
    , s.$$FCTCOL_006 s$$FCTCOL_006, t.$$FCTCOL_006 t$$FCTCOL_006
    , s.$$FCTCOL_007 s$$FCTCOL_007, t.$$FCTCOL_007 t$$FCTCOL_007
    , s.$$FCTCOL_008 s$$FCTCOL_008, t.$$FCTCOL_008 t$$FCTCOL_008
    , s.$$FCTCOL_009 s$$FCTCOL_009, t.$$FCTCOL_009 t$$FCTCOL_009
    , s.$$FCTCOL_010 s$$FCTCOL_010, t.$$FCTCOL_010 t$$FCTCOL_010
    , s.$$FCTCOL_011 s$$FCTCOL_011, t.$$FCTCOL_011 t$$FCTCOL_011
    , s.$$FCTCOL_012 s$$FCTCOL_012, t.$$FCTCOL_012 t$$FCTCOL_012
    , s.$$FCTCOL_013 s$$FCTCOL_013, t.$$FCTCOL_013 t$$FCTCOL_013
    , s.$$FCTCOL_014 s$$FCTCOL_014, t.$$FCTCOL_014 t$$FCTCOL_014
    , s.$$FCTCOL_015 s$$FCTCOL_015, t.$$FCTCOL_015 t$$FCTCOL_015
    , s.$$FCTCOL_016 s$$FCTCOL_016, t.$$FCTCOL_016 t$$FCTCOL_016
    , s.$$FCTCOL_017 s$$FCTCOL_017, t.$$FCTCOL_017 t$$FCTCOL_017
    , s.$$FCTCOL_018 s$$FCTCOL_018, t.$$FCTCOL_018 t$$FCTCOL_018
    , s.$$FCTCOL_019 s$$FCTCOL_019, t.$$FCTCOL_019 t$$FCTCOL_019
    , s.$$FCTCOL_020 s$$FCTCOL_020, t.$$FCTCOL_020 t$$FCTCOL_020
    , s.$$FCTCOL_021 s$$FCTCOL_021, t.$$FCTCOL_021 t$$FCTCOL_021
    , s.$$FCTCOL_022 s$$FCTCOL_022, t.$$FCTCOL_022 t$$FCTCOL_022
    , s.$$FCTCOL_023 s$$FCTCOL_023, t.$$FCTCOL_023 t$$FCTCOL_023
    , s.$$FCTCOL_024 s$$FCTCOL_024, t.$$FCTCOL_024 t$$FCTCOL_024

$$SELECT INTO_BODY[$$FCTTBL[]_NFD]
FROM
    $$FCTTBL[]_MFL s, $$FCTTBL[]_MFL t
WHERE
    s.iss = t.iss AND s.ss_key = t.ss_key
AND
    s.date_key = (SELECT MAX(date_key) FROM $$FCTTBL[]_MFL u WHERE
    u.iss = s.iss AND u.ss_key = s.ss_key AND u.date_key < t.date_key)

--#BLOCK_END# MakeNFD

```

```

/*****
--
-- Insert BOOKs for deltas with same dim keys
--
-- If the dimensions don't change then we create a
-- new booking order (as long as at least one of the facts
-- have changed)
--
-- IDM: Insert Delta More
--
*****/

--#BLOCK_BEGIN# MakeIDM

$$SELECT INTO_BEGIN[$$FCTTBL[]_IDM]
SELECT
    tiss iss,
    tss_key ss_key,
    tdate_key date_key,
    ttranstype_key transtype_key,
    0 seq
    , t$$DIMKEYR_01 t$$DIMKEYR_01
    , t$$DIMKEYR_02 t$$DIMKEYR_02
    , t$$DIMKEYR_03 t$$DIMKEYR_03
    , t$$DIMKEYR_04 t$$DIMKEYR_04
    , t$$DIMKEYR_05 t$$DIMKEYR_05
    , t$$DIMKEYR_06 t$$DIMKEYR_06
    , t$$DIMKEYR_07 t$$DIMKEYR_07
    , t$$DIMKEYR_08 t$$DIMKEYR_08
    , t$$DIMKEYR_09 t$$DIMKEYR_09
    , t$$DIMKEYR_10 t$$DIMKEYR_10
    , t$$DEGKEY_01 t$$DEGKEY_01
    , t$$DEGKEY_02 t$$DEGKEY_02
    , t$$DEGKEY_03 t$$DEGKEY_03
    ,
    t$$FCTCOL_001-s$$FCTCOL_001 t$$FCTCOL_001
    , t$$FCTCOL_002-s$$FCTCOL_002 t$$FCTCOL_002
    , t$$FCTCOL_003-s$$FCTCOL_003 t$$FCTCOL_003
    , t$$FCTCOL_004-s$$FCTCOL_004 t$$FCTCOL_004
    , t$$FCTCOL_005-s$$FCTCOL_005 t$$FCTCOL_005
    , t$$FCTCOL_006-s$$FCTCOL_006 t$$FCTCOL_006
    , t$$FCTCOL_007-s$$FCTCOL_007 t$$FCTCOL_007
    , t$$FCTCOL_008-s$$FCTCOL_008 t$$FCTCOL_008
    , t$$FCTCOL_009-s$$FCTCOL_009 t$$FCTCOL_009
    , t$$FCTCOL_010-s$$FCTCOL_010 t$$FCTCOL_010
    , t$$FCTCOL_011-s$$FCTCOL_011 t$$FCTCOL_011
    , t$$FCTCOL_012-s$$FCTCOL_012 t$$FCTCOL_012
    , t$$FCTCOL_013-s$$FCTCOL_013 t$$FCTCOL_013
    , t$$FCTCOL_014-s$$FCTCOL_014 t$$FCTCOL_014
    , t$$FCTCOL_015-s$$FCTCOL_015 t$$FCTCOL_015
    , t$$FCTCOL_016-s$$FCTCOL_016 t$$FCTCOL_016
    , t$$FCTCOL_017-s$$FCTCOL_017 t$$FCTCOL_017
    , t$$FCTCOL_018-s$$FCTCOL_018 t$$FCTCOL_018
    , t$$FCTCOL_019-s$$FCTCOL_019 t$$FCTCOL_019
    , t$$FCTCOL_020-s$$FCTCOL_020 t$$FCTCOL_020
    , t$$FCTCOL_021-s$$FCTCOL_021 t$$FCTCOL_021
    , t$$FCTCOL_022-s$$FCTCOL_022 t$$FCTCOL_022
    , t$$FCTCOL_023-s$$FCTCOL_023 t$$FCTCOL_023
    , t$$FCTCOL_024-s$$FCTCOL_024 t$$FCTCOL_024

$$SELECT INTO_BODY[$$FCTTBL[]_IDM]
FROM
    $$FCTTBL[]_NFD d
WHERE
    (
        (s$$DIMKEYR_06 = t$$DIMKEYR_06) AND
        (s$$DIMKEYR_05 = t$$DIMKEYR_05) AND
        (s$$DIMKEYR_07 = t$$DIMKEYR_07) AND
        (s$$DIMKEYR_04 = t$$DIMKEYR_04) AND
        (s$$DIMKEYR_08 = t$$DIMKEYR_08) AND
        (s$$DIMKEYR_03 = t$$DIMKEYR_03) AND
    )

```



```

(s$$$DIMKEYR_09 = t$$$DIMKEYR_09) AND
(s$$$DIMKEYR_02 = t$$$DIMKEYR_02) AND
(s$$$DIMKEYR_10 = t$$$DIMKEYR_10) AND
(s$$$DIMKEYR_01 = t$$$DIMKEYR_01)
)

```

AND

```

(
(s$$$FCTCOL_001 <> t$$$FCTCOL_001)
OR (s$$$FCTCOL_002 <> t$$$FCTCOL_002)
OR (s$$$FCTCOL_003 <> t$$$FCTCOL_003)
OR (s$$$FCTCOL_004 <> t$$$FCTCOL_004)
OR (s$$$FCTCOL_005 <> t$$$FCTCOL_005)
OR (s$$$FCTCOL_006 <> t$$$FCTCOL_006)
OR (s$$$FCTCOL_007 <> t$$$FCTCOL_007)
OR (s$$$FCTCOL_008 <> t$$$FCTCOL_008)
OR (s$$$FCTCOL_009 <> t$$$FCTCOL_009)
OR (s$$$FCTCOL_010 <> t$$$FCTCOL_010)
OR (s$$$FCTCOL_011 <> t$$$FCTCOL_011)
OR (s$$$FCTCOL_012 <> t$$$FCTCOL_012)
OR (s$$$FCTCOL_013 <> t$$$FCTCOL_013)
OR (s$$$FCTCOL_014 <> t$$$FCTCOL_014)
OR (s$$$FCTCOL_015 <> t$$$FCTCOL_015)
OR (s$$$FCTCOL_016 <> t$$$FCTCOL_016)
OR (s$$$FCTCOL_017 <> t$$$FCTCOL_017)
OR (s$$$FCTCOL_018 <> t$$$FCTCOL_018)
OR (s$$$FCTCOL_019 <> t$$$FCTCOL_019)
OR (s$$$FCTCOL_020 <> t$$$FCTCOL_020)
OR (s$$$FCTCOL_021 <> t$$$FCTCOL_021)
OR (s$$$FCTCOL_022 <> t$$$FCTCOL_022)
OR (s$$$FCTCOL_023 <> t$$$FCTCOL_023)
OR (s$$$FCTCOL_024 <> t$$$FCTCOL_024)
)

```

--#BLOCK_END# MakeIDM

```

/*****
--
-- Insert negative BOOKs for deltas with different dim keys
--
-- If one of the dimensions change then we first create a lose transaction for
-- all the previous facts. (Negate all the facts from the earlier of the two
-- transactions)
--
-- ILM: Insert Lost More
--
*****/

```

--#BLOCK_BEGIN# MakeILM

```

$$$SELECT INTO BEGIN[$$FCTTBL[]_ILM]
SELECT
    siss iss,
    sss_key ss_key,
    tdate_key date_key,
    stranstype_key transtype_key,
    0 seq
,
    s$$$DIMKEYR_01 $$$DIMKEYR_01
,
    s$$$DIMKEYR_02 $$$DIMKEYR_02
,
    s$$$DIMKEYR_03 $$$DIMKEYR_03
,
    s$$$DIMKEYR_04 $$$DIMKEYR_04
,
    s$$$DIMKEYR_05 $$$DIMKEYR_05
,
    s$$$DIMKEYR_06 $$$DIMKEYR_06
,
    s$$$DIMKEYR_07 $$$DIMKEYR_07
,
    s$$$DIMKEYR_08 $$$DIMKEYR_08
,
    s$$$DIMKEYR_09 $$$DIMKEYR_09
,
    s$$$DIMKEYR_10 $$$DIMKEYR_10
,
    s$$$DEGKEY_01 $$$DEGKEY_01
,
    s$$$DEGKEY_02 $$$DEGKEY_02
,
    s$$$DEGKEY_03 $$$DEGKEY_03
,
    -s$$$FCTCOL_001 $$$FCTCOL_001

```

```

, -$$$FCTCOL_002 $$$FCTCOL_002
, -$$$FCTCOL_003 $$$FCTCOL_003
, -$$$FCTCOL_004 $$$FCTCOL_004
, -$$$FCTCOL_005 $$$FCTCOL_005
, -$$$FCTCOL_006 $$$FCTCOL_006
, -$$$FCTCOL_007 $$$FCTCOL_007
, -$$$FCTCOL_008 $$$FCTCOL_008
, -$$$FCTCOL_009 $$$FCTCOL_009
, -$$$FCTCOL_010 $$$FCTCOL_010
, -$$$FCTCOL_011 $$$FCTCOL_011
, -$$$FCTCOL_012 $$$FCTCOL_012
, -$$$FCTCOL_013 $$$FCTCOL_013
, -$$$FCTCOL_014 $$$FCTCOL_014
, -$$$FCTCOL_015 $$$FCTCOL_015
, -$$$FCTCOL_016 $$$FCTCOL_016
, -$$$FCTCOL_017 $$$FCTCOL_017
, -$$$FCTCOL_018 $$$FCTCOL_018
, -$$$FCTCOL_019 $$$FCTCOL_019
, -$$$FCTCOL_020 $$$FCTCOL_020
, -$$$FCTCOL_021 $$$FCTCOL_021
, -$$$FCTCOL_022 $$$FCTCOL_022
, -$$$FCTCOL_023 $$$FCTCOL_023
, -$$$FCTCOL_024 $$$FCTCOL_024

```

```

$$SELECT INTO BODY[$$FCTBL[]_ILM]
FROM

```

```

    $$FCTBL[]_NFD d

```

```

WHERE

```

```

(
(s$$DIMKEYR_06 <> t$$DIMKEYR_06) OR
(s$$DIMKEYR_05 <> t$$DIMKEYR_05) OR
(s$$DIMKEYR_07 <> t$$DIMKEYR_07) OR
(s$$DIMKEYR_04 <> t$$DIMKEYR_04) OR
(s$$DIMKEYR_08 <> t$$DIMKEYR_08) OR
(s$$DIMKEYR_03 <> t$$DIMKEYR_03) OR
(s$$DIMKEYR_09 <> t$$DIMKEYR_09) OR
(s$$DIMKEYR_02 <> t$$DIMKEYR_02) OR
(s$$DIMKEYR_10 <> t$$DIMKEYR_10) OR
(s$$DIMKEYR_01 <> t$$DIMKEYR_01)
)

```

```

AND

```

```

(
(s$$$FCTCOL_001 <> 0)
OR
(s$$$FCTCOL_002 <> 0)
OR
(s$$$FCTCOL_003 <> 0)
OR
(s$$$FCTCOL_004 <> 0)
OR
(s$$$FCTCOL_005 <> 0)
OR
(s$$$FCTCOL_006 <> 0)
OR
(s$$$FCTCOL_007 <> 0)
OR
(s$$$FCTCOL_008 <> 0)
OR
(s$$$FCTCOL_009 <> 0)
OR
(s$$$FCTCOL_010 <> 0)
OR
(s$$$FCTCOL_011 <> 0)
OR
(s$$$FCTCOL_012 <> 0)
OR
(s$$$FCTCOL_013 <> 0)
OR
(s$$$FCTCOL_014 <> 0)
OR
(s$$$FCTCOL_015 <> 0)
OR
(s$$$FCTCOL_016 <> 0)
OR
(s$$$FCTCOL_017 <> 0)
OR
(s$$$FCTCOL_018 <> 0)
OR
(s$$$FCTCOL_019 <> 0)
OR
(s$$$FCTCOL_020 <> 0)
OR
(s$$$FCTCOL_021 <> 0)
OR
(s$$$FCTCOL_022 <> 0)
OR
(s$$$FCTCOL_023 <> 0)
OR
(s$$$FCTCOL_024 <> 0)
)

```

```

--#BLOCK_END# MakeILM

```

```

/*****

```

```

--
-- Insert BOOKS for deltas with different dim keys
--
-- When a dimension key changes then we can simply insert all the new facts with the
-- new dimension keys
--
-- Note that seq = 1 here because this is the second transaction on this date for
-- this order.
--
-- IRM: Insert Rebook More
--
/*****/

--#BLOCK_BEGIN# MakeIRM

$$SELECT INTO BEGIN[$$FCTTBL[]_IRM]
SELECT
    tiss iss,
    tss_key ss_key,
    tdate_key date_key,
    ttranstype_key transtype_key,
    1 seq
    , t$$DIMKEYR_01 t$$DIMKEYR_01
    , t$$DIMKEYR_02 t$$DIMKEYR_02
    , t$$DIMKEYR_03 t$$DIMKEYR_03
    , t$$DIMKEYR_04 t$$DIMKEYR_04
    , t$$DIMKEYR_05 t$$DIMKEYR_05
    , t$$DIMKEYR_06 t$$DIMKEYR_06
    , t$$DIMKEYR_07 t$$DIMKEYR_07
    , t$$DIMKEYR_08 t$$DIMKEYR_08
    , t$$DIMKEYR_09 t$$DIMKEYR_09
    , t$$DIMKEYR_10 t$$DIMKEYR_10
    , t$$DEGKEY_01 t$$DEGKEY_01
    , t$$DEGKEY_02 t$$DEGKEY_02
    , t$$DEGKEY_03 t$$DEGKEY_03
    ,
    t$$FCTCOL_001 t$$FCTCOL_001
    , t$$FCTCOL_002 t$$FCTCOL_002
    , t$$FCTCOL_003 t$$FCTCOL_003
    , t$$FCTCOL_004 t$$FCTCOL_004
    , t$$FCTCOL_005 t$$FCTCOL_005
    , t$$FCTCOL_006 t$$FCTCOL_006
    , t$$FCTCOL_007 t$$FCTCOL_007
    , t$$FCTCOL_008 t$$FCTCOL_008
    , t$$FCTCOL_009 t$$FCTCOL_009
    , t$$FCTCOL_010 t$$FCTCOL_010
    , t$$FCTCOL_011 t$$FCTCOL_011
    , t$$FCTCOL_012 t$$FCTCOL_012
    , t$$FCTCOL_013 t$$FCTCOL_013
    , t$$FCTCOL_014 t$$FCTCOL_014
    , t$$FCTCOL_015 t$$FCTCOL_015
    , t$$FCTCOL_016 t$$FCTCOL_016
    , t$$FCTCOL_017 t$$FCTCOL_017
    , t$$FCTCOL_018 t$$FCTCOL_018
    , t$$FCTCOL_019 t$$FCTCOL_019
    , t$$FCTCOL_020 t$$FCTCOL_020
    , t$$FCTCOL_021 t$$FCTCOL_021
    , t$$FCTCOL_022 t$$FCTCOL_022
    , t$$FCTCOL_023 t$$FCTCOL_023
    , t$$FCTCOL_024 t$$FCTCOL_024

$$SELECT INTO BODY[$$FCTTBL[]_IRM]
FROM
    $$FCTTBL[]_NFD d
WHERE
    (
        (s$$DIMKEYR_06 <> t$$DIMKEYR_06) OR
        (s$$DIMKEYR_05 <> t$$DIMKEYR_05) OR
        (s$$DIMKEYR_07 <> t$$DIMKEYR_07) OR
        (s$$DIMKEYR_04 <> t$$DIMKEYR_04) OR
        (s$$DIMKEYR_08 <> t$$DIMKEYR_08) OR
    )

```

```

(s$$$DIMKEYR_03 <> t$$$DIMKEYR_03) OR
(s$$$DIMKEYR_09 <> t$$$DIMKEYR_09) OR
(s$$$DIMKEYR_02 <> t$$$DIMKEYR_02) OR
(s$$$DIMKEYR_10 <> t$$$DIMKEYR_10) OR
(s$$$DIMKEYR_01 <> t$$$DIMKEYR_01)
)
AND
(
(t$$$FCTCOL_001 <> 0)
OR
(t$$$FCTCOL_002 <> 0)
OR
(t$$$FCTCOL_003 <> 0)
OR
(t$$$FCTCOL_004 <> 0)
OR
(t$$$FCTCOL_005 <> 0)
OR
(t$$$FCTCOL_006 <> 0)
OR
(t$$$FCTCOL_007 <> 0)
OR
(t$$$FCTCOL_008 <> 0)
OR
(t$$$FCTCOL_009 <> 0)
OR
(t$$$FCTCOL_010 <> 0)
OR
(t$$$FCTCOL_011 <> 0)
OR
(t$$$FCTCOL_012 <> 0)
OR
(t$$$FCTCOL_013 <> 0)
OR
(t$$$FCTCOL_014 <> 0)
OR
(t$$$FCTCOL_015 <> 0)
OR
(t$$$FCTCOL_016 <> 0)
OR
(t$$$FCTCOL_017 <> 0)
OR
(t$$$FCTCOL_018 <> 0)
OR
(t$$$FCTCOL_019 <> 0)
OR
(t$$$FCTCOL_020 <> 0)
OR
(t$$$FCTCOL_021 <> 0)
OR
(t$$$FCTCOL_022 <> 0)
OR
(t$$$FCTCOL_023 <> 0)
OR
(t$$$FCTCOL_024 <> 0)
)

--#BLOCK_END# MakeIRM

/*****
-- Delete the output tables
*****/

--#BLOCK_BEGIN# DropOutput

$$$$DDL_BEGIN
$$$$DROP_TABLE IF EXISTS[$$FCTTBL[]$$NEXT]
$$$$DROP_TABLE IF EXISTS[$$FCTTBL[]_INC]
$$$$DDL_END

--#BLOCK_END# DropOutput

/*****
--Create FC table in case force_close was
-- not run
*****/

--#BLOCK_BEGIN# MakeFC

DECLARE $$VAR[fc_exists] $$EPIINT$$EOS

$$$$DDL_BEGIN_NO_DECLARE

$$VAR_ASSIGN_BEGIN[fc_exists]
SELECT COUNT(1)
$$VAR_ASSIGN_INT0[fc_exists]
FROM $$$QLSERVER[sysobjects]$$ORACLE[tabs]
WHERE
$$$QLSERVER[id = object_id('dbo.'$$FCTTBL[]_FC') AND sysstat & 0xf = 3]
$$ORACLE[table_name = UPPER('$$FCTTBL[]_FC')]
$$VAR_ASSIGN_END

$$IF[$$VAR[fc_exists] = 0]
$$$$DDL_EXEC[

```

```

$$SELECT INTO BEGIN[$$FCTTBL[]_FC]
SELECT
    *
$$SELECT INTO BODY[$$FCTTBL[]_FC]
FROM
    $$FCTTBL[]$$CURR
WHERE
    1=0
]
$$END_IF
$$DDL_END

--#BLOCK_END# MakeFC

/*****
-- Create the incremental table
*****/

--#BLOCK_BEGIN# MakeINC

$$SELECT INTO BEGIN[$$FCTTBL[]_INC]
SELECT
    *
$$SELECT INTO BODY[$$FCTTBL[]_INC]
FROM $$FCTTBL[] TIN UNION ALL
SELECT * FROM $$FCTTBL[]_IL UNION ALL
SELECT * FROM $$FCTTBL[]_IR UNION ALL
SELECT * FROM $$FCTTBL[]_IRD UNION ALL
SELECT * FROM $$FCTTBL[]_IND UNION ALL
SELECT * FROM $$FCTTBL[]_IRM UNION ALL
SELECT * FROM $$FCTTBL[]_ILM UNION ALL
SELECT * FROM $$FCTTBL[]_FC UNION ALL
SELECT * FROM $$FCTTBL[]_IDM

--#BLOCK_END# MakeINC

/*****
-- CR158: We want to load _IMI table and still keep the non-descending
-- order so that the clustered index on a fact table can be created
-- without sorting. This way can speed up significantly in creating a
-- clustered index on a very large already sorted fact table.
*****/

--#BLOCK_BEGIN# MakeIMI

$$SELECT INTO BEGIN[$$FCTTBL[]_IMI]
SELECT
    *
$$SELECT INTO BODY[$$FCTTBL[]_IMI]
FROM $$FCTTBL[]$$CURR
WHERE date_key >= (SELECT MIN(date_key) FROM $$FCTTBL[]_INC)
UNION ALL
SELECT * FROM $$FCTTBL[]_INC
$$SQLSERVER[ORDER BY
    date_key
    ,   $$DIMKEYR_01
    ,   $$DIMKEYR_02
    ,   $$DIMKEYR_03
    ,   $$DIMKEYR_04
    ,   $$DIMKEYR_05
    ,   $$DIMKEYR_06
    ,   $$DIMKEYR_07
    ,   $$DIMKEYR_08
    ,   $$DIMKEYR_09
    ,   $$DIMKEYR_10
]

--#BLOCK_END# MakeIMI

/*****
-- Create the new fact table and incremental table
*****/

```

```

--
-- Note that transaction tables must be built before
-- these statements are run
/*****

--#BLOCK_BEGIN# MakeNewFact

$$SELECT INTO_BEGIN[$$FCTTBL[]$$NEXT]
SELECT *
$$SELECT INTO_BODY[$$FCTTBL[]$$NEXT]
FROM $$FCTTBL[]$$CURR s
WHERE s.date_key < (SELECT MIN(date_key) FROM $$FCTTBL[]_INC)
UNION ALL
SELECT * FROM $$FCTTBL[]_IMI

--#BLOCK_END# MakeNewFact

/*****
-- Count processed, inserted rows
/*****

--#BLOCK_BEGIN# SPResults

DECLARE $$VAR[count_INC] $$EPIINT$$EOS

BEGIN

$$VAR_ASSIGN_BEGIN[count_INC]
SELECT COUNT(1)
$$VAR_ASSIGN_INTO[count_INC]
FROM $$FCTTBL[]_INC
$$VAR_ASSIGN_END

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM $$FCTTBL[]_MFL$$EOS

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', $$VAR[count_INC] - COUNT(1) FROM $$FCTTBL[]_TIN$$EOS

END$$EOS

--#BLOCK_END# SPResults

/*****
-- Set join order for SQL Server
/*****

--#BLOCK_BEGIN# ForcePlanOff

$$SQLSERVER[SET FORCEPLAN OFF]

--#BLOCK_END# ForcePlanOff

/*****
-- Drop temp tables and TXN and TIN table
/*****

--#BLOCK_BEGIN# DropTempsAfter

$$DDL_BEGIN
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_TIN]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_TMI]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_FC]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_TXN]
$$DROP_TABLE_IF_EXISTS[Concat_MFL]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_1ST]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IL]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IR]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IRD]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IND]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_NFD]

```

```

$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IRM]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IDM]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_ILM]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_IMI]
$$DDL_END

--#BLOCK_END# DropTempsAfter

--#TEMPLATE_END# load_state
--#TEMPLATE_BEGIN# load_trans

/*****/
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- load_trans
--
-- Move transaction-like staging data into Fact table - create a temp
-- table with TXN extension that has all old rows along with new rows.
-- Also produce a TIN (TXN INC) table that has only the new rows
--
-- Note that the new table will also include all existing rows from
-- the Fact table.
--
/*****/
/*****/
-- Delete output tables
--
-- Output table is called TXN and includes old and new rows
--
-- Also, leave around _TIN as incremental table from this
-- procedure
--
-- We also create a table called _TMI which contains all the
-- _TIN records plus the records of overlapping period from the
-- old existing fact table.
/*****/
--#BLOCK_BEGIN# RemoveOutput

$$DDL_BEGIN
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_TXN]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_TMI]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_TIN]
$$DDL_END

--#BLOCK_END# RemoveOutput

/*****/
-- Set join order for SQL Server
/*****/

--#BLOCK_BEGIN# ForcePlanOn

$$SQLSERVER[SET FORCEPLAN ON]

--#BLOCK_END# ForcePlanOn

/*****/
-- Remove stuff already in fact table
--
-- Note that currently this filter implies that once a transactional
-- fact entry is made it cannot be changed - and no further fact
-- entries on that date or any previous date can be made either
/*****/

--#BLOCK_BEGIN# CreateTIN

$$SELECT_INTO_BEGIN[$$FCTTBL[]_TIN]
SELECT

```

```

s.iss,
s.ss_key,
s.date_key,
s.transtype_key,
s.ikey seq
, s.$$DIMKEYR_01
, s.$$DIMKEYR_02
, s.$$DIMKEYR_03
, s.$$DIMKEYR_04
, s.$$DIMKEYR_05
, s.$$DIMKEYR_06
, s.$$DIMKEYR_07
, s.$$DIMKEYR_08
, s.$$DIMKEYR_09
, s.$$DIMKEYR_10
, s.$$DEGKEY_01
, s.$$DEGKEY_02
, s.$$DEGKEY_03
,
s.$$FCTCOL_001
, s.$$FCTCOL_002
, s.$$FCTCOL_003
, s.$$FCTCOL_004
, s.$$FCTCOL_005
, s.$$FCTCOL_006
, s.$$FCTCOL_007
, s.$$FCTCOL_008
, s.$$FCTCOL_009
, s.$$FCTCOL_010
, s.$$FCTCOL_011
, s.$$FCTCOL_012
, s.$$FCTCOL_013
, s.$$FCTCOL_014
, s.$$FCTCOL_015
, s.$$FCTCOL_016
, s.$$FCTCOL_017
, s.$$FCTCOL_018
, s.$$FCTCOL_019
, s.$$FCTCOL_020
, s.$$FCTCOL_021
, s.$$FCTCOL_022
, s.$$FCTCOL_023
, s.$$FCTCOL_024
,
$$SELECT INTO_BODY[$$FCTTBL[]_TIN]
FROM
    $$FSTGTBL[]_MAP s, bus_process b
WHERE
    NOT EXISTS (SELECT * FROM $$FCTTBL[]$$CURR f WHERE
        s.iss = f.iss AND
        s.ss_key = f.ss_key AND
        f.date_key >= s.date_key)
AND
(
    (s.$$FCTCOL_001 <> 0)
OR
    (s.$$FCTCOL_002 <> 0)
OR
    (s.$$FCTCOL_003 <> 0)
OR
    (s.$$FCTCOL_004 <> 0)
OR
    (s.$$FCTCOL_005 <> 0)
OR
    (s.$$FCTCOL_006 <> 0)
OR
    (s.$$FCTCOL_007 <> 0)
OR
    (s.$$FCTCOL_008 <> 0)
OR
    (s.$$FCTCOL_009 <> 0)
OR
    (s.$$FCTCOL_010 <> 0)
OR
    (s.$$FCTCOL_011 <> 0)
OR
    (s.$$FCTCOL_012 <> 0)
OR
    (s.$$FCTCOL_013 <> 0)
OR
    (s.$$FCTCOL_014 <> 0)
OR
    (s.$$FCTCOL_015 <> 0)
OR
    (s.$$FCTCOL_016 <> 0)
OR
    (s.$$FCTCOL_017 <> 0)
OR
    (s.$$FCTCOL_018 <> 0)

```



```

OR      (s.$$FCTCOL_019 <> 0)
OR      (s.$$FCTCOL_020 <> 0)
OR      (s.$$FCTCOL_021 <> 0)
OR      (s.$$FCTCOL_022 <> 0)
OR      (s.$$FCTCOL_023 <> 0)
OR      (s.$$FCTCOL_024 <> 0)
}
AND
      s.process_key = b.process_key AND b.process_name = 'LoadTrans'

--#BLOCK_END# CreateTIN

/*****
-- Set join order for SQL Server
*****/

--#BLOCK_BEGIN# ForcePlanOff

$$$SQLSERVER[SET FORCEPLAN OFF]

--#BLOCK_END# ForcePlanOff

/*****
-- CR158: We want to load _TMI table and still keep the non-descending
-- order so that the clustered index on a fact table can be created
-- without sorting. This way can speed up significantly in creating a
-- clustered index on a very large already sorted fact table.
*****/

--#BLOCK_BEGIN# CreateTMI

$$$SELECT_INTO_BEGIN[$$FCTTBL[]_TMI]
SELECT
      *
$$$SELECT_INTO_BODY[$$FCTTBL[]_TMI]
FROM
      $$FCTTBL[]$SCURR
WHERE
      date_key >= (SELECT MAX(date_key) FROM $$FCTTBL[]_TIN)
UNION ALL
SELECT
      *
FROM
      $$FCTTBL[]_TIN
$$$SQLSERVER[ORDER BY
      date_key
      ,   $$DIMKEYR_01
      ,   $$DIMKEYR_02
      ,   $$DIMKEYR_03
      ,   $$DIMKEYR_04
      ,   $$DIMKEYR_05
      ,   $$DIMKEYR_06
      ,   $$DIMKEYR_07
      ,   $$DIMKEYR_08
      ,   $$DIMKEYR_09
      ,   $$DIMKEYR_10
]

--#BLOCK_END# CreateTMI

/*****
-- Insert everything into the new fact table
*****/

--#BLOCK_BEGIN# CreateTXN

$$$SELECT_INTO_BEGIN[$$FCTTBL[]_TXN]
SELECT
      *
$$$SELECT_INTO_BODY[$$FCTTBL[]_TXN]
FROM

```

```

        $$FCTTBL[]$$CURR s
WHERE s.date_key < (SELECT MAX(date_key) FROM $$FCTTBL[]_TIN)
UNION ALL
SELECT
    *
FROM
    $$FCTTBL[]_TMI f

--#BLOCK_END# CreateTXN

/*****/
-- Count inserted data and put results into communication table
/*****/

--#BLOCK_BEGIN# SPResults

BEGIN

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM $$FSTGTBL[]_MAP$$EOS

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', COUNT(1) FROM $$FCTTBL[]_TIN$$EOS

END$$EOS

--#BLOCK_END# SPResults

--#TEMPLATE_END# load_trans

--#TEMPLATE_BEGIN# index_fact

/*****/
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- Post processing after an extraction run
--
-- Reindex fact tables
-- CR158: added WITH SORTED_DATA in creating cluster index on fact table
--
-- Remove any temp tables generated during the extraction
--
/*****/

/*****/
-- Primary key index the fact table
/*****/

--#BLOCK_BEGIN# PKIndexFact

$$DDL_BEGIN
$$DDL_EXEC[

CREATE UNIQUE INDEX XPK$$FCTTBL[]$$NEXT ON $$FCTTBL[]$$NEXT
(
    iss , ss_key , date_key , transtype_key , seq
)
]
$$DDL_END

--#BLOCK_END# PKIndexFact

/*****/
-- Inversion index the fact table
/*****/

--#BLOCK_BEGIN# IEIndexFact

$$DDL_BEGIN
$$DDL_EXEC[

```

```

CREATE $$SQLSERVER[CLUSTERED ] INDEX XIEK$$FCTTBL[]$$NEXT ON $$FCTTBL[]$$NEXT
(
    date_key
    , $$DIMKEYR_01
    , $$DIMKEYR_02
    , $$DIMKEYR_03
    , $$DIMKEYR_04
    , $$DIMKEYR_05
    , $$DIMKEYR_06
    , $$DIMKEYR_07
    , $$DIMKEYR_08
    , $$DIMKEYR_09
    , $$DIMKEYR_10
) $$SQLSERVER[WITH SORTED_DATA]

]

$$DDL_END

--#BLOCK_END# IEIndexFact

/*****/
-- Remove any mapped tables
/*****/

--#BLOCK_BEGIN# RemoveTemps

$$DDL_BEGIN
$$DROP_TABLE_IF_EXISTS[$$FSTGTBL[]_MAP]
$$DDL_END

--#BLOCK_END# RemoveTemps

--#TEMPLATE_END# index_fact
--#TEMPLATE_BEGIN# ren_trans

/*****/
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- ren_trans
--
-- Epiphany Marketing Software, 1997
--
-- Simply change the name of the transaction new table to the
-- actual fact table name - used for Fact tables that don't have
-- any stored procedure other than load_trans attached to them
--
/*****/

/*****/
-- Delete the output tables
/*****/

--#BLOCK_BEGIN# RemoveOutput

$$DDL_BEGIN
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]$$NEXT]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_INC]
$$DDL_END

--#BLOCK_END# RemoveOutput

/*****/
-- Move all transaction rows into the correct new fact table
-- name. Note that we would use sp_rename, except it
-- doesn't work with DB name prefixes
--
-- TBD: Rename instead of re-select
/*****/

```

```

--#BLOCK_BEGIN# BuildNewFact

$$SELECT_INTO_BEGIN[$$FCTTBL[]$$NEXT]
SELECT
*
$$SELECT_INTO_BODY[$$FCTTBL[]$$NEXT]
FROM
    $$FCTTBL[]_TXN

--#BLOCK_END# BuildNewFact

/*****/
-- Preserve incremental table
/*****/

--#BLOCK_BEGIN# BuildIncremental

$$SELECT_INTO_BEGIN[$$FCTTBL[]_INC]
SELECT
*
$$SELECT_INTO_BODY[$$FCTTBL[]_INC]
FROM
    $$FCTTBL[]_TIN

--#BLOCK_END# BuildIncremental

/*****/
-- Count inserted data and put results into communication table
/*****/

--#BLOCK_BEGIN# SPResults

BEGIN

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM $$FCTTBL[]_TXN$$EOS

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', COUNT(1) FROM $$FCTTBL[]_TXN$$EOS

END$$EOS

--#BLOCK_END# SPResults

/*****/
-- Remove temp tables
/*****/

--#BLOCK_BEGIN# RemoveTemps

$$DDL_BEGIN
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_TXN]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_TIN]
$$DROP_TABLE_IF_EXISTS[$$FCTTBL[]_TMI]
$$DDL_END

--#BLOCK_END# RemoveTemps

--#TEMPLATE_END# ren_trans

--#TEMPLATE_BEGIN# map_keys

/*****/
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- map_keys
--
-- Epiphany Marketing Software
--

```

```

-- Map dimension keys from Staging table and report
-- on unjoined rows
--
/*****/

/*****/
-- Remove output table
/*****/

--#BLOCK_BEGIN# DropTemp

$$DDL_BEGIN
$$DROP TABLE_IF_EXISTS[$$FSTGTBL[]_MAP]
$$DDL_END

--#BLOCK_END# DropTemp

/*****/
-- Set join order for SQL Server
/*****/

--#BLOCK_BEGIN# ForcePlanOn

$$$SQLSERVER[SET FORCEPLAN ON]

--#BLOCK_END# ForcePlanOn

/*****/
-- Map dimension keys via Inner joins
/*****/

--#BLOCK_BEGIN# MapAll

$$$SELECT INTO_BEGIN[$$FSTGTBL[]_MAP]
SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    s.ikey,
    s.process_key
    $$PIPE_STATE
,
    m_04.$$DIMKEY_04 $$DIMKEYR_04
,
    m_03.$$DIMKEY_03 $$DIMKEYR_03
,
    m_06.$$DIMKEY_06 $$DIMKEYR_06
,
    m_02.$$DIMKEY_02 $$DIMKEYR_02
,
    m_08.$$DIMKEY_08 $$DIMKEYR_08
,
    m_05.$$DIMKEY_05 $$DIMKEYR_05
,
    m_09.$$DIMKEY_09 $$DIMKEYR_09
,
    m_01.$$DIMKEY_01 $$DIMKEYR_01
,
    m_07.$$DIMKEY_07 $$DIMKEYR_07
,
    m_10.$$DIMKEY_10 $$DIMKEYR_10
,
    $$DEGKEY_03
,
    $$DEGKEY_02
,
    $$DEGKEY_01
,
    s.$$FCTCOL_001
,
    s.$$FCTCOL_002
,
    s.$$FCTCOL_003
,
    s.$$FCTCOL_004
,
    s.$$FCTCOL_005
,
    s.$$FCTCOL_006
,
    s.$$FCTCOL_007
,
    s.$$FCTCOL_008
,
    s.$$FCTCOL_009
,
    s.$$FCTCOL_010
,
    s.$$FCTCOL_011
,
    s.$$FCTCOL_012
,
    s.$$FCTCOL_013

```

```

, s.$$FCTCOL_014
, s.$$FCTCOL_015
, s.$$FCTCOL_016
, s.$$FCTCOL_017
, s.$$FCTCOL_018
, s.$$FCTCOL_019
, s.$$FCTCOL_020
, s.$$FCTCOL_021
, s.$$FCTCOL_022
, s.$$FCTCOL_023
, s.$$FCTCOL_024

```

```

$$SELECT INTO BODY[$$FSTGTBL[]_MAP]
FROM

```

```

    $$FSTGTBL[] s
,   $$MAPTBL_04$$NEXT m_04 $$SQLSERVER[(index = 1)]
,   $$MAPTBL_03$$NEXT m_03 $$SQLSERVER[(index = 1)]
,   $$MAPTBL_06$$NEXT m_06 $$SQLSERVER[(index = 1)]
,   $$MAPTBL_02$$NEXT m_02 $$SQLSERVER[(index = 1)]
,   $$MAPTBL_08$$NEXT m_08 $$SQLSERVER[(index = 1)]
,   $$MAPTBL_05$$NEXT m_05 $$SQLSERVER[(index = 1)]
,   $$MAPTBL_09$$NEXT m_09 $$SQLSERVER[(index = 1)]
,   $$MAPTBL_01$$NEXT m_01 $$SQLSERVER[(index = 1)]
,   $$MAPTBL_07$$NEXT m_07 $$SQLSERVER[(index = 1)]
,   $$MAPTBL_10$$NEXT m_10 $$SQLSERVER[(index = 1)]

```

```

WHERE 1=1
AND m_04.iss = s.iss AND m_04.$$DSTGKEY_04 = s.$$DSTGKEYR_04
AND m_03.iss = s.iss AND m_03.$$DSTGKEY_03 = s.$$DSTGKEYR_03
AND m_06.iss = s.iss AND m_06.$$DSTGKEY_06 = s.$$DSTGKEYR_06
AND m_02.iss = s.iss AND m_02.$$DSTGKEY_02 = s.$$DSTGKEYR_02
AND m_08.iss = s.iss AND m_08.$$DSTGKEY_08 = s.$$DSTGKEYR_08
AND m_05.iss = s.iss AND m_05.$$DSTGKEY_05 = s.$$DSTGKEYR_05
AND m_09.iss = s.iss AND m_09.$$DSTGKEY_09 = s.$$DSTGKEYR_09
AND m_01.iss = s.iss AND m_01.$$DSTGKEY_01 = s.$$DSTGKEYR_01
AND m_07.iss = s.iss AND m_07.$$DSTGKEY_07 = s.$$DSTGKEYR_07
AND m_10.iss = s.iss AND m_10.$$DSTGKEY_10 = s.$$DSTGKEYR_10

```

```

--#BLOCK_END# MapAll

```

```

/*****
-- Set join order for SQL Server
*****/

```

```

--#BLOCK_BEGIN# ForcePlanOff

```

```

$$SQLSERVER[SET FORCEPLAN OFF]

```

```

--#BLOCK_END# ForcePlanOff

```

```

/*****
-- Look for unjoined data, Report on processed rows
*****/

```

```

--#BLOCK_BEGIN# SPResults

```

```

$$DECLARE BEGIN
$$DECLARE_BODY[$$VAR[unjoined] $$EPIINT]
$$DECLARE_BODY[$$VAR[processed] $$EPIINT]

```

```

BEGIN

```

```

$$VAR ASSIGN BEGIN[processed]
SELECT COUNT(1)
$$VAR ASSIGN INTO[processed]
FROM $$FSTGTBL[]
$$VAR ASSIGN_END

```

```

$$VAR ASSIGN BEGIN[unjoined]
SELECT $$VAR[processed] - COUNT(1)
$$VAR ASSIGN INTO[unjoined]

```

```

FROM $$FSTGTBL[]_MAP
$$VAR_ASSIGN_END

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'UNJOINED', $$VAR[unjoined] $$NO_FROM_LIST$$EOS

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', $$VAR[processed] $$NO_FROM_LIST$$EOS

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', $$VAR[processed] - $$VAR[unjoined] $$NO_FROM_LIST$$EOS

ENDS$$EOS

--#BLOCK_END# SPResults

/*****
-- Index this temp table
*****/

--#BLOCK_BEGIN# IndexMap

$$DDL_BEGIN
$$DDL_EXEC[
CREATE INDEX X$$FSTGTBL[]_MAP ON $$FSTGTBL[]_MAP
(
    iss, ss_key, date_key, ikey
)
]
$$DDL_END

--#BLOCK_END# IndexMap

--#TEMPLATE_END# map_keys
--#TEMPLATE_BEGIN# upd_unj

/*****
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved
--
-- upd_unj
--
-- Epiphany Marketing Software
--
-- Update all dimension keys to 'UNKNOWN' in staging table
-- where referential integrity fails
--
*****/

/*****
-- Count the number of rows to update in the staging table - that is, those
-- that have at least one Foreign key where referential integrity fails
*****/

--#BLOCK_BEGIN# CountUnj

BEGIN

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM $$FSTGTBL[]$$EOS

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'MODIFIED', COUNT(1)
FROM
    $$FSTGTBL[] s
WHERE 1=0
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_04$$NEXT m_04 WHERE m_04.iss = s.iss AND
m_04.$$DSTGKEY_04 = $$DSTGKEYR_04)
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_03$$NEXT m_03 WHERE m_03.iss = s.iss AND
m_03.$$DSTGKEY_03 = $$DSTGKEYR_03)
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_06$$NEXT m_06 WHERE m_06.iss = s.iss AND

```

```

m_06.$$DSTGKEY_06 = $$DSTGKEYR_06)
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_02$$NEXT m_02 WHERE m_02.iss = s.iss AND
m_02.$$DSTGKEY_02 = $$DSTGKEYR_02)
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_08$$NEXT m_08 WHERE m_08.iss = s.iss AND
m_08.$$DSTGKEY_08 = $$DSTGKEYR_08)
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_05$$NEXT m_05 WHERE m_05.iss = s.iss AND
m_05.$$DSTGKEY_05 = $$DSTGKEYR_05)
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_09$$NEXT m_09 WHERE m_09.iss = s.iss AND
m_09.$$DSTGKEY_09 = $$DSTGKEYR_09)
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_01$$NEXT m_01 WHERE m_01.iss = s.iss AND
m_01.$$DSTGKEY_01 = $$DSTGKEYR_01)
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_07$$NEXT m_07 WHERE m_07.iss = s.iss AND
m_07.$$DSTGKEY_07 = $$DSTGKEYR_07)
OR NOT EXISTS (SELECT 1 FROM $$MAPTBL_10$$NEXT m_10 WHERE m_10.iss = s.iss AND
m_10.$$DSTGKEY_10 = $$DSTGKEYR_10)
$$EOS
END$$EOS

```

```
--#BLOCK_END# CountUnj
```

```

/*****/
-- Update foreign keys where referential integrity fails
/*****/

```

```
--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_04
```

```

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_04 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_04$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_04 = $$FSTGTBL[].$$DSTGKEYR_04)

```

```
--#BLOCK_END# UpdateUnj$$DSTGKEYR_04
```

```
--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_03
```

```

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_03 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_03$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_03 = $$FSTGTBL[].$$DSTGKEYR_03)

```

```
--#BLOCK_END# UpdateUnj$$DSTGKEYR_03
```

```
--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_06
```

```

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_06 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_06$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_06 = $$FSTGTBL[].$$DSTGKEYR_06)

```

```
--#BLOCK_END# UpdateUnj$$DSTGKEYR_06
```

```
--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_02
```

```

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_02 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_02$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_02 = $$FSTGTBL[].$$DSTGKEYR_02)

```

```
--#BLOCK_END# UpdateUnj$$DSTGKEYR_02
```

```
--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_08
```

```

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_08 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_08$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_08 = $$FSTGTBL[].$$DSTGKEYR_08)

```

```
--#BLOCK_END# UpdateUnj$$DSTGKEYR_08
```

```
--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_05
```

```

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_05 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_05$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_05 = $$FSTGTBL[].$$DSTGKEYR_05)

```

```
--#BLOCK_END# UpdateUnj$$DSTGKEYR_05
```



```

--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_09

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_09 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_09$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_09 = $$FSTGTBL[].$$DSTGKEYR_09)

--#BLOCK_END# UpdateUnj$$DSTGKEYR_09

--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_01

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_01 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_01$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_01 = $$FSTGTBL[].$$DSTGKEYR_01)

--#BLOCK_END# UpdateUnj$$DSTGKEYR_01

--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_07

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_07 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_07$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_07 = $$FSTGTBL[].$$DSTGKEYR_07)

--#BLOCK_END# UpdateUnj$$DSTGKEYR_07

--#BLOCK_BEGIN# UpdateUnj$$DSTGKEYR_10

UPDATE $$FSTGTBL[] SET $$DSTGKEYR_10 = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM $$MAPTBL_10$$NEXT m
WHERE m.iss = $$FSTGTBL[].iss AND m.$$DSTGKEY_10 = $$FSTGTBL[].$$DSTGKEYR_10)

--#BLOCK_END# UpdateUnj$$DSTGKEYR_10

--#TEMPLATE_END# 'upd_unj

```

The following are the post-parsed SQL source for the adaptive templates as filled in with corresponding schema definitions.

```

--#TEMPLATE_BEGIN# force_close

/*****
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- force_close
--
-- Close out deleted orders - those that no longer appear in the
-- staging table
--
-- SEE SAFETY VALVE BELOW
--
*****/

/*****
-- Delete temporary tables

```

```

/*****/
--#BLOCK_BEGIN# DropTemps

IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_FC') AND sysstat & 0xf =
3) DROP TABLE Order_0_FC

--#BLOCK_END# DropTemps

/*****/
-- Insert negative BOOKs for deleted orders
--
-- FC: ForceClose
/*****/

--#BLOCK_BEGIN# MakeFC

SELECT
    f.iss,
    f.ss_key,
    MAX(f.date_key) date_key,
    MIN(f.transtype_key) transtype_key,
    MAX(f.seq) + 1 seq
,
    f.customerbillto_key
,
    f.product_key
,
    f.application_key
,
    f.program_key
,
    f.customershipto_key
,
    f.territory_key
,
    f.warehouse_key
,
    -SUM(f.net_price) net_price
,
    -SUM(f.number_units) number_units

INTO Order_0_FC
FROM
    Order_0_A f
WHERE
    NOT EXISTS
    (SELECT 1 FROM OrderStage_MAP s WHERE s.iss = f.iss AND s.ss_key = f.ss_key)
GROUP BY
    f.iss,
    f.ss_key
,
    f.customerbillto_key
,
    f.product_key
,
    f.application_key
,
    f.program_key
,
    f.customershipto_key
,
    f.territory_key
,
    f.warehouse_key

HAVING
    (
        (SUM(f.net_price) <> 0)
        OR
        (SUM(f.number_units) <> 0)
    )
AND
    MIN(f.transtype_key) <= 99
AND
    MIN(f.transtype_key) >= 1

--#BLOCK_END# MakeFC

/*****/
-- SAFETY VALVE - THIS PROC ONLY DOES ANYTHING
-- IF THE STAGING TABLE HAS AT LEAST ONE ROW
/*****/

```

```

--#BLOCK_BEGIN# SafetyValue

DECLARE @count_MAP INT

BEGIN

SELECT @count_MAP = (
SELECT COUNT(1)

FROM OrderStage_MAP
)

IF (@count_MAP = 0)
DELETE FROM Order_0_FC

END

--#BLOCK_END# SafetyValue

/*****
-- Count processed, inserted rows
*****/

--#BLOCK_BEGIN# SPResults

BEGIN

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM Order_0_A

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', COUNT(1) FROM Order_0_FC

END

--#BLOCK_END# SPResults

--#TEMPLATE_END# force_close

--#TEMPLATE_BEGIN# load_state

/*****
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- load_state
--
-- Load order bookings into fact table by creating transactional
-- data from state data
--
-- load_trans must be run before this procedure to create TIN table
--
*****/

/*****
-- Delete temporary tables
*****/

--#BLOCK_BEGIN# DropTemps

IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_MFL') AND sysstat & 0xf
= 3) DROP TABLE Order_0_MFL
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_1ST') AND sysstat & 0xf
= 3) DROP TABLE Order_0_1ST
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IL') AND sysstat & 0xf =
3) DROP TABLE Order_0_IL
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IR') AND sysstat & 0xf =
3) DROP TABLE Order_0_IR

```

```

IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IRD') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IRD
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IND') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IND
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_NFD') AND sysstat & 0xf
= 3) DROP TABLE Order_0_NFD
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IRM') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IRM
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IDM') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IDM
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_ILM') AND sysstat & 0xf
= 3) DROP TABLE Order_0_ILM
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IMI') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IMI

--#BLOCK_END# DropTemps

/*****/
-- Set join order for SQL Server
/*****/

--#BLOCK_BEGIN# ForcePlanOn

SET FORCEPLAN ON

--#BLOCK_END# ForcePlanOn

/*****/
-- Remove rows older than fact table - history can not be rewritten - only
-- the last date for an order can be changed. Note that we compare transtype's
-- because SHIP type transactions might occur at a later date and we don't want
-- those to interfere
--
-- Also, since the staging table may have multiple entries for a given order on
-- a single day - we assume that the last one inserted in the Staging table will
-- be used (since ikey is an IDENTITY column)
--
-- Note that a given ss_key must use the same Booking transtype for all of time,
-- otherwise the transtype_key
--
-- MFL: Mapped Filtered
/*****/

--#BLOCK_BEGIN# MakeMFL

SELECT
    s.*
INTO Order_0_MFL
FROM
    OrderStage_MAP s, bus_process b
WHERE
    ((s.date_key >= (SELECT MAX(date_key) FROM Order_0_A f WHERE
        s.iss = f.iss AND s.ss_key = f.ss_key AND
        s.transtype_key = f.transtype_key))
    OR NOT EXISTS (SELECT * FROM Order_0_A f WHERE
        s.iss = f.iss AND s.ss_key = f.ss_key AND
        s.transtype_key = f.transtype_key))
AND
    s.ikey = (SELECT MAX(t.ikey) FROM OrderStage_MAP t WHERE
        s.iss = t.iss AND
        s.ss_key = t.ss_key AND
        s.date_key = t.date_key AND
        t.process_key = b.process_key)
AND
    s.process_key = b.process_key AND b.process_name = 'LoadState'

--#BLOCK_END# MakeMFL

/*****/
-- Index MFL table for later queries

```

```

/*****/
--#BLOCK_BEGIN# IndexMFL

EXEC('
CREATE INDEX XOrder_0_MFL ON Order_0_MFL
(
    iss, ss_key, date_key
)
')

--#BLOCK_END# IndexMFL

/*****/
-- Get oldest state rows for each unique sskey
--
-- We need to treat the first entry for each order
-- in the staging table separately from all others, since
-- only the first entry needs to be compared with
-- already existing fact entry rows to create transactions.
-- All subsequent dates for that order in the Fact table
-- can be delta'd with other staging table entries - see the
-- section below on Pairwise deltas.
--
-- MFL should be indexed
--
-- 1ST: The first record for each iss, ss_key
/*****/

--#BLOCK_BEGIN# Make1ST

SELECT
    s.*
INTO Order_0_1ST
FROM
    Order_0_MFL s
WHERE
    s.date_key = (SELECT MIN(date_key) FROM Order_0_MFL t WHERE
        s.iss = t.iss AND s.ss_key = t.ss_key)

--#BLOCK_END# Make1ST

/*****/
-- Index 1ST for later queries
/*****/

--#BLOCK_BEGIN# Index1ST

EXEC('
CREATE UNIQUE INDEX XPKOrder_0_1ST ON Order_0_1ST
(
    iss, ss_key
)
')

--#BLOCK_END# Index1ST

/*****/
-- Insert negative BOOKs for changed dim keys
--
-- This query will add up all existing Books and Loss's
-- for this order and the net facts will be cancelled out
-- with the old Dimension keys. Note that an invariant of this
-- procedure is that only one set of dimensions at a time
-- can have non-zero facts.
--

```

```

-- Fact table Should be indexed
--
-- HAVING Clause is needed to prevent changing of dimensions
-- on fully shipped order from causing a transaction - no sense
-- creating fact rows with all zero's in them
--
-- Note that we increment the sequence number just in case
-- this new transaction occurs on the same date as the last
-- existing one in the fact table - to avoid index errors
--
-- IL: InsertLost
/*****
--#BLOCK_BEGIN# MakeIL

SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    MAX(f.seq) + 1 seq
    ,
    f.customerbillto_key
    ,
    f.product_key
    ,
    f.application_key
    ,
    f.program_key
    ,
    f.customershipto_key
    ,
    f.territory_key
    ,
    f.warehouse_key
    ,
    -SUM(f.net_price) net_price
    ,
    -SUM(f.number_units) number_units

INTO Order_0_IL
FROM
    Order_0_1ST s, Order_0_A f

WHERE
    s.iss = f.iss AND s.ss_key = f.ss_key

AND
    ((s.territory_key <> f.territory_key) OR
    (s.customershipto_key <> f.customershipto_key) OR
    (s.warehouse_key <> f.warehouse_key) OR
    (s.program_key <> f.program_key) OR
    (s.application_key <> f.application_key) OR
    (s.product_key <> f.product_key) OR
    (s.customerbillto_key <> f.customerbillto_key) )

GROUP BY
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key
    ,
    f.customerbillto_key
    ,
    f.product_key
    ,
    f.application_key
    ,
    f.program_key
    ,
    f.customershipto_key
    ,
    f.territory_key
    ,
    f.warehouse_key

HAVING
    MIN(f.transtype_key) = s.transtype_key

AND
    (
    {SUM(f.net_price) <> 0}
    OR
    {SUM(f.number_units) <> 0}
    )

--#BLOCK_END# MakeIL

/****
-- Index IL for later queries

```

```

/*****
--#BLOCK_BEGIN# IndexIL

EXEC('
CREATE INDEX XPKOrder_0_IL ON Order_0_IL
(
    iss, ss_key
)
')

--#BLOCK_END# IndexIL

/*****
-- Insert BOOKs for changed dim keys
--
-- When a dimension changes then just create a booking
-- transaction for whatever we negated above with the new
-- dimension and fact values
--
-- 1ST should be indexed
--
-- Note that we add one to whatever we used as the last
-- seq because this transaction occurs on the same
-- date as the negative one above
--
-- IR: Insert Rebook
/*****

--#BLOCK_BEGIN# MakeIR

SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    l.transtype_key,
    l.seq + 1 seq
    ,
    s.customerbillto_key
    ,
    s.product_key
    ,
    s.application_key
    ,
    s.program_key
    ,
    s.customershipto_key
    ,
    s.territory_key
    ,
    s.warehouse_key
    ,
    -l.net_price net_price
    ,
    -l.number_units number_units

INTO Order_0_IR
FROM
    Order_0_IL l, Order_0_1ST s
WHERE l.iss = s.iss AND l.ss_key = s.ss_key

--#BLOCK_END# MakeIR

/*****
-- Insert BOOKs for changed dim keys where fact
-- also changed
--
-- When a dimension changes at the same time as
-- a fact then we need to make up the fact difference
--
-- 1ST should be indexed
--
-- Note that we add two to whatever we used as the last
-- seq because this transaction occurs on the same
-- date as the negative and positive ones above
--
--

```

```

-- Note also that the Left Outer join uses transtype_key
-- so that only the Bookings at the old value will be counted.
-- Whereas above for the negative transaction value
-- we want to include Shipments in our calculation, here
-- we only want to see how Booking Facts have changed.
--
-- Here again, only one Booking transaction type is supported
-- per ss_key
--
-- IRD: Insert Rebook delta
/*****

--#BLOCK_BEGIN# MakeIRD

SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    l.seq + 2 seq
    , s.customerbillto_key
    , s.product_key
    , s.application_key
    , s.program_key
    , s.customershipto_key
    , s.territory_key
    , s.warehouse_key
    , MAX(s.net_price)-ISNULL(SUM(f.net_price) , 0) net_price
    , MAX(s.number_units)-ISNULL(SUM(f.number_units) , 0) number_units

INTO Order_0_IRD
FROM
    Order_0_IL l, Order_0_1ST s
    LEFT OUTER JOIN Order_0_A f ON s.iss = f.iss AND s.ss_key = f.ss_key AND
    s.transtype_key = f.transtype_key
WHERE
    l.iss = s.iss AND l.ss_key = s.ss_key

GROUP BY
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    l.seq
    , s.customerbillto_key
    , s.product_key
    , s.application_key
    , s.program_key
    , s.customershipto_key
    , s.territory_key
    , s.warehouse_key

HAVING
    (ISNULL(SUM(f.net_price) , 0) <> MAX(s.net_price))
OR
    (ISNULL(SUM(f.number_units) , 0) <> MAX(s.number_units))

--#BLOCK_END# MakeIRD

/*****/
-- Insert BOOKS for deltas with same dim keys OR for
-- brand new orders.
--
-- Note that we DON'T want to count Shipments
-- (so shipment ss_key's should be different from
-- order ss_keys) since we just want bookings to sum up
-- to whatever this transaction says they should be.
--
-- Fact table should be indexed
--

```



```

-- WHERE clause prevents double booking on changed
-- dimension - if we didn't use the NOT EXISTS clause
-- then this query would repeat the work of the last one
-- above - which we have already taken care of
--
-- HAVING clause ensures that multiple 0 records don't
-- get inserted whenever this procedure is run
--
-- Note that we increment the sequence number just in case
-- this new transaction occurs on the same date as the last
-- existing one in the fact table - to avoid index errors
--
-- IND: Insert New Delta
/*****
--#BLOCK_BEGIN# MakeIND

SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    ISNULL(MAX(f.seq) , 0) + 1 seq
    ,
    s.customerbillto_key
    ,
    s.product_key
    ,
    s.application_key
    ,
    s.program_key
    ,
    s.customershipto_key
    ,
    s.territory_key
    ,
    s.warehouse_key
    ,
    MAX(s.net_price)-ISNULL(SUM(f.net_price) , 0) net_price
    ,
    MAX(s.number_units)-ISNULL(SUM(f.number_units) , 0) number_units

INTO Order_0_IND
FROM
    Order_0_1ST s LEFT OUTER JOIN Order_0_A f ON
        s.iss = f.iss AND s.ss_key = f.ss_key AND f.transtype_key = s.transtype_key
WHERE
    NOT EXISTS (SELECT * FROM Order_0_IL WHERE iss = s.iss AND ss_key = s.ss_key)

GROUP BY
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key
    ,
    s.customerbillto_key
    ,
    s.product_key
    ,
    s.application_key
    ,
    s.program_key
    ,
    s.customershipto_key
    ,
    s.territory_key
    ,
    s.warehouse_key

HAVING
    (ISNULL(SUM(f.net_price) , 0) <> MAX(s.net_price))
    OR
    (ISNULL(SUM(f.number_units) , 0) <> MAX(s.number_units))

--#BLOCK_END# MakeIND

/*****/
-- Form pairwise deltas for all rows except earliest for each sskey
--
-- Each row created in NFD will consist of two sequential entries from the
-- staing table. So if N enties for an order exist in MFL (after we have filtered
-- out same-date duplicates) then all the queries above will deal with the earliest entry,
whereas
-- all the queries below (including this one) will deal with the N-1 deltaing transactions
--

```

```

-- This query assumes that MFL will already have been filtered
-- to have a single record for each order/datekey
--
-- NFD: Not First Delta
/*****/

--#BLOCK_BEGIN# MakeNFD

SELECT
    s.iss siss, t.iss tiss
  ,   s.ss_key sss_key, t.ss_key tss_key
  ,   s.date_key sdate_key, t.date_key tdate_key
  ,   s.transtype_key stranstype_key, t.transtype_key ttranstype_key
  ,   s.customerbillto_key scustomerbillto_key, t.customerbillto_key tcustomerbillto_key
  ,   s.product_key sproduct_key, t.product_key tproduct_key
  ,   s.application_key sapplication_key, t.application_key tapplication_key
  ,   s.program_key sprogram_key, t.program_key tprogram_key
  ,   s.customershipto_key scustomershipto_key, t.customershipto_key tcustomershipto_key
  ,   s.territory_key sterritory_key, t.territory_key tterritory_key
  ,   s.warehouse_key swarehouse_key, t.warehouse_key twarehouse_key
  ,   s.net_price snet_price, t.net_price tnet_price
  ,   s.number_units snumber_units, t.number_units tnumber_units

INTO Order_0_NFD
FROM
    Order_0_MFL s, Order_0_MFL t
WHERE
    s.iss = t.iss AND s.ss_key = t.ss_key
AND
    s.date_key = (SELECT MAX(date_key) FROM Order_0_MFL u WHERE
    u.iss = s.iss AND u.ss_key = s.ss_key AND u.date_key < t.date_key)

--#BLOCK_END# MakeNFD

/*****/
--
-- Insert BOOKs for deltas with same dim keys
--
-- If the dimensions don't change then we create a
-- new booking order (as long as at least one of the facts
-- have changed)
--
-- IDM: Insert Delta More
--
/*****/

--#BLOCK_BEGIN# MakeIDM

SELECT
    tiss iss,
    tss_key ss_key,
    tdate_key date_key,
    ttranstype_key transtype_key,
    0 seq
  ,   tcustomerbillto_key customerbillto_key
  ,   tproduct_key product_key
  ,   tapplication_key application_key
  ,   tprogram_key program_key
  ,   tcustomershipto_key customershipto_key
  ,   tterritory_key territory_key
  ,   twarehouse_key warehouse_key
  ,
  ,   tnet_price-snet_price net_price
  ,   tnumber_units-snumber_units number_units

INTO Order_0_IDM
FROM
    Order_0_NFD d
WHERE

```

```

        (
            (territory_key = tterritory_key) AND
            (scustomershipto_key = tcustomershipto_key) AND
            (swarehouse_key = twarehouse_key) AND
            (sprogram_key = tprogram_key) AND
            (sapplication_key = tapplication_key) AND
            (sproduct_key = tproduct_key) AND
            (scustomerbillto_key = tcustomerbillto_key)
        )
    AND
        (
            (snet_price <> tnet_price)
        OR
            (snumber_units <> tnumber_units)
        )

--#BLOCK_END# MakeIDM

/*****
--
-- Insert negative BOOKs for deltas with different dim keys
--
-- If one of the dimensions change then we first create a lose transaction for
-- all the previous facts. (Negate all the facts from the earlier of the two
-- transactions)
--
-- ILM: Insert Lost More
--
*****/

--#BLOCK_BEGIN# MakeILM

SELECT
    siss iss,
    sss_key ss_key,
    tdate_key date_key,
    stranstype_key transtype_key,
    0 seq
    ,
    scustomerbillto_key customerbillto_key
    ,
    sproduct_key product_key
    ,
    sapplication_key application_key
    ,
    sprogram_key program_key
    ,
    scustomershipto_key customershipto_key
    ,
    sterritory_key territory_key
    ,
    swarehouse_key warehouse_key
    ,
    -snet_price net_price
    ,
    -snumber_units number_units

INTO Order_0_ILM
FROM
    Order_0_NFD d
WHERE
    (
        (territory_key <> tterritory_key) OR
        (scustomershipto_key <> tcustomershipto_key) OR
        (swarehouse_key <> twarehouse_key) OR
        (sprogram_key <> tprogram_key) OR
        (sapplication_key <> tapplication_key) OR
        (sproduct_key <> tproduct_key) OR
        (scustomerbillto_key <> tcustomerbillto_key)
    )
    AND
        (
            (snet_price <> 0)
        OR
            (snumber_units <> 0)
        )

--#BLOCK_END# MakeILM

*****/

```

```

--
-- Insert BOOKs for deltas with different dim keys
--
-- When a dimension key changes then we can simply insert all the new facts with the
-- new dimension keys
--
-- Note that seq = 1 here because this is the second transaction on this date for
-- this order.
--
-- IRM: Insert Rebook More
--
/*****/

--#BLOCK_BEGIN# MakeIRM

SELECT
    tiss iss,
    tss key ss_key,
    tdate_key date_key,
    ttranstype_key transtype_key,
    1 seq
,
    tcustomerbillto_key customerbillto_key
,
    tproduct_key product_key
,
    tapplication_key application_key
,
    tprogram_key program_key
,
    tcustomershipto_key customershipto_key
,
    tterritory_key territory_key
,
    twarehouse_key warehouse_key

,
    tnet_price net_price
,
    tnumber_units number_units

INTO Order_0_IRM
FROM
    Order_0_NFD d
WHERE
    (
        (sterterritory_key <> tterritory_key) OR
        (scustomershipto_key <> tcustomershipto_key) OR
        (swarehouse_key <> twarehouse_key) OR
        (sprogram_key <> tprogram_key) OR
        (sapplication_key <> tapplication_key) OR
        (sproduct_key <> tproduct_key) OR
        (scustomerbillto_key <> tcustomerbillto_key)
    )
AND
    (
        (tnet_price <> 0)
    )
OR
    (
        (tnumber_units <> 0)
    )

--#BLOCK_END# MakeIRM

/*****/
-- Delete the output tables
/*****/

--#BLOCK_BEGIN# DropOutput

IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_B') AND sysstat & 0xf =
3) DROP TABLE Order_0_B
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_INC') AND sysstat & 0xf
= 3) DROP TABLE Order_0_INC

--#BLOCK_END# DropOutput

/*****/
--Create FC table in case force close was

```

```

-- not run
/*****

--#BLOCK_BEGIN# MakeFC

DECLARE @fc_exists INT

SELECT @fc_exists = (
SELECT COUNT(1)

FROM sysobjects
WHERE
id = object_id('dbo.Order_0_FC') AND sysstat & 0xf = 3
)

IF (@fc_exists = 0)
EXEC('

SELECT
*
INTO Order_0_FC
FROM
    Order_0_A
WHERE
    1=0
')

--#BLOCK_END# MakeFC

/*****
-- Create the incremental table
/*****

--#BLOCK_BEGIN# MakeINC

SELECT
*
INTO Order_0_INC
FROM Order_0_TIN UNION ALL
SELECT * FROM Order_0_IL UNION ALL
SELECT * FROM Order_0_IR UNION ALL
SELECT * FROM Order_0_IRD UNION ALL
SELECT * FROM Order_0_IND UNION ALL
SELECT * FROM Order_0_IRM UNION ALL
SELECT * FROM Order_0_ILM UNION ALL
SELECT * FROM Order_0_FC UNION ALL
SELECT * FROM Order_0_IDM

--#BLOCK_END# MakeINC

/*****
-- CR158: We want to load _IMI table and still keep the non-descending
-- order so that the clustered index on a fact table can be created
-- without sorting. This way can speed up significantly in creating a
-- clustered index on a very large already sorted fact table.
/*****

--#BLOCK_BEGIN# MakeIMI

SELECT
*
INTO Order_0_IMI
FROM Order_0_A
WHERE date_key >= (SELECT MIN(date_key) FROM Order_0_INC)

```

```

UNION ALL
SELECT * FROM Order_0_INC
ORDER BY
    date_key
    , customerbillto_key
    , product_key
    , application_key
    , program_key
    , customershipto_key
    , territory_key
    , warehouse_key

--#BLOCK_END# MakeIMI

/*****
-- Create the new fact table and incremental table
--
-- Note that transaction tables must be built before
-- these statements are run
*****/

--#BLOCK_BEGIN# MakeNewFact

SELECT *
INTO Order_0_B
FROM Order_0_A s
WHERE s.date_key < (SELECT MIN(date_key) FROM Order_0_INC)
UNION ALL
SELECT * FROM Order_0_IMI

--#BLOCK_END# MakeNewFact

/*****
-- Count processed, inserted rows
*****/

--#BLOCK_BEGIN# SPResults

DECLARE @count_INC INT

BEGIN

SELECT @count_INC = (
SELECT COUNT(1)

FROM Order_0_INC
)

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM Order_0_MFL

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', @count_INC - COUNT(1) FROM Order_0_TIN

END

--#BLOCK_END# SPResults

/*****
-- Set join order for SQL Server
*****/

--#BLOCK_BEGIN# ForcePlanOff

SET FORCEPLAN OFF

--#BLOCK_END# ForcePlanOff

/*****

```

```
-- Drop temp tables and TXN and TIN table
/*****/

--#BLOCK_BEGIN# DropTempsAfter

IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_TIN') AND sysstat & 0xf
= 3) DROP TABLE Order_0_TIN
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_TMI') AND sysstat & 0xf
= 3) DROP TABLE Order_0_TMI
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_FC') AND sysstat & 0xf =
3) DROP TABLE Order_0_FC
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_TXN') AND sysstat & 0xf
= 3) DROP TABLE Order_0_TXN
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Concat_MFL') AND sysstat & 0xf =
3) DROP TABLE Concat_MFL
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_1ST') AND sysstat & 0xf
= 3) DROP TABLE Order_0_1ST
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IL') AND sysstat & 0xf =
3) DROP TABLE Order_0_IL
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IR') AND sysstat & 0xf =
3) DROP TABLE Order_0_IR
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IRD') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IRD
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IND') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IND
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_NFD') AND sysstat & 0xf
= 3) DROP TABLE Order_0_NFD
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IRM') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IRM
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IDM') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IDM
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_ILM') AND sysstat & 0xf
= 3) DROP TABLE Order_0_ILM
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_IMI') AND sysstat & 0xf
= 3) DROP TABLE Order_0_IMI

--#BLOCK_END# DropTempsAfter

--#TEMPLATE_END# load_state
--#TEMPLATE_BEGIN# load_trans

/*****/
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- load_trans
--
-- Move transaction-like staging data into Fact table - create a temp
-- table with TXN extension that has all old rows along with new rows.
-- Also produce a TIN (TXN INC) table that has only the new rows
--
-- Note that the new table will also include all existing rows from
-- the Fact table.
--
/*****/

/*****/
-- Delete output tables
--
-- Output table is called TXN and includes old and new rows
--
-- Also, leave around _TIN as incremental table from this
-- procedure
--
-- We also create a table called _TMI which contains all the
-- _TIN records plus the records of overlapping period from the
-- old existing fact table.
/*****/
```



```

--#BLOCK_BEGIN# ForcePlanOff

SET FORCEPLAN OFF

--#BLOCK_END# ForcePlanOff

/*****
-- CR158: We want to load _TMI table and still keep the non-descending
-- order so that the clustered index on a fact table can be created
-- without sorting. This way can speed up significantly in creating a
-- clustered index on a very large already sorted fact table.
*****/

--#BLOCK_BEGIN# CreateTMI

SELECT
    *
INTO Order_0_TMI
FROM
    Order_0_A
WHERE
    date_key >= (SELECT MAX(date_key) FROM Order_0_TIN)
UNION ALL
SELECT
    *
FROM
    Order_0_TIN
ORDER BY
    date_key
    , customerbillto_key
    , product_key
    , application_key
    , program_key
    , customershipto_key
    , territory_key
    , warehouse_key

--#BLOCK_END# CreateTMI

/*****
-- Insert everything into the new fact table
*****/

--#BLOCK_BEGIN# CreateTXN

SELECT
    *
INTO Order_0_TXN
FROM
    Order_0_A s
WHERE s.date_key < (SELECT MAX(date_key) FROM Order_0_TIN)
UNION ALL
SELECT
    *
FROM
    Order_0_TMI f

--#BLOCK_END# CreateTXN

/*****
-- Count inserted data and put results into communication table
*****/

--#BLOCK_BEGIN# SPResults

BEGIN

```

```

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM OrderStage_MAP

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', COUNT(1) FROM Order_0_TIN

END

--#BLOCK_END# SPResults

--#TEMPLATE_END# load_trans

--#TEMPLATE_BEGIN# index_fact

/*****/
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- Post processing after an extraction run
--
-- Reindex fact tables
-- CR158: added WITH SORTED_DATA in creating cluster index on fact table
--
-- Remove any temp tables generated during the extraction
--
/*****/

/*****/
-- Primary key index the fact table
/*****/

--#BLOCK_BEGIN# PKIndexFact

EXEC('

CREATE UNIQUE INDEX XPKOrder_0_B ON Order_0_B
(
    iss , ss_key , date_key , transtype_key , seq
)
')

--#BLOCK_END# PKIndexFact

/*****/
-- Inversion index the fact table
/*****/

--#BLOCK_BEGIN# IEIndexFact

EXEC('

CREATE CLUSTERED INDEX XIEKOrder_0_B ON Order_0_B
(
    date_key
    , customerbillto_key
    , product_key
    , application_key
    , program_key
    , customershipto_key
    , territory_key
    , warehouse_key
) WITH SORTED_DATA
')

--#BLOCK_END# IEIndexFact

```

```

/*****
-- Remove any mapped tables
/*****/

--#BLOCK_BEGIN# RemoveTemps

IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.OrderStage_MAP') AND sysstat & 0xf = 3) DROP TABLE OrderStage_MAP

--#BLOCK_END# RemoveTemps

--#TEMPLATE_END# index_fact
--#TEMPLATE_BEGIN# ren_trans

/*****/
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- ren_trans
--
-- Epiphany Marketing Software, 1997
--
-- Simply change the name of the transaction new table to the
-- actual fact table name - used for Fact tables that don't have
-- any stored procedure other than load_trans attached to them
--
/*****/

/*****/
-- Delete the output tables
/*****/

--#BLOCK_BEGIN# RemoveOutput

IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_B') AND sysstat & 0xf = 3) DROP TABLE Order_0_B
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_INC') AND sysstat & 0xf = 3) DROP TABLE Order_0_INC

--#BLOCK_END# RemoveOutput

/*****/
-- Move all transaction rows into the correct new fact table
-- name. Note that we would use sp_rename, except it
-- doesn't work with DB name prefixes
--
-- TBD: Rename instead of re-select
/*****/

--#BLOCK_BEGIN# BuildNewFact

SELECT
    *
INTO Order_0_B
FROM
    Order_0_TXN

--#BLOCK_END# BuildNewFact

/*****/
-- Preserve incremental table
/*****/

--#BLOCK_BEGIN# BuildIncremental

```

```

SELECT
    *
INTO Order_0_INC
FROM
    Order_0_TIN

--#BLOCK_END# BuildIncremental

/*****/
-- Count inserted data and put results into communication table
/*****/

--#BLOCK_BEGIN# SPResults

BEGIN

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM Order_0_TXN

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', COUNT(1) FROM Order_0_TXN

END

--#BLOCK_END# SPResults

/*****/
-- Remove temp tables
/*****/

--#BLOCK_BEGIN# RemoveTemps

IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_TXN') AND sysstat & 0xf
= 3) DROP TABLE Order_0_TXN
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_TIN') AND sysstat & 0xf
= 3) DROP TABLE Order_0_TIN
IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.Order_0_TMI') AND sysstat & 0xf
= 3) DROP TABLE Order_0_TMI

--#BLOCK_END# RemoveTemps

--#TEMPLATE_END# ren_trans

--#TEMPLATE_BEGIN# map_keys

/*****/
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved.
--
-- map_keys
--
-- Epiphany Marketing Software
--
-- Map dimension keys from Staging table and report
-- on unjoined rows
--
/*****/

/*****/
-- Remove output table
/*****/

--#BLOCK_BEGIN# DropTemp

IF EXISTS (SELECT 1 FROM sysobjects WHERE id = object_id('dbo.OrderStage_MAP') AND sysstat &
0xf = 3) DROP TABLE OrderStage_MAP

```

```

--#BLOCK_END# DropTemp

/*****/
-- Set join order for SQL Server
/*****/

--#BLOCK_BEGIN# ForcePlanOn

SET FORCEPLAN ON

--#BLOCK_END# ForcePlanOn

/*****/
-- Map dimension keys via Inner joins
/*****/

--#BLOCK_BEGIN# MapAll

SELECT
    s.iss,
    s.ss_key,
    s.date_key,
    s.transtype_key,
    s.ikey,
    s.process_key
,
    m_04.program_key program_key
,
    m_03.application_key application_key
,
    m_06.territory_key territory_key
,
    m_02.product_key product_key
,
    m_05.customer_key customershipto_key
,
    m_01.customer_key customerbillto_key
,
    m_07.warehouse_key warehouse_key
,
    s.net_price
,
    s.number_units

INTO OrderStage_MAP
FROM
    OrderStage s
,
    ProgramMap_B m_04 (index = 1)
,
    ApplicationMap_B m_03 (index = 1)
,
    TerritoryMap_B m_06 (index = 1)
,
    ProductMap_B m_02 (index = 1)
,
    CustomerMap_B m_05 (index = 1)
,
    CustomerMap_B m_01 (index = 1)
,
    WarehouseMap_B m_07 (index = 1)

WHERE 1=1
AND m_04.iss = s.iss AND m_04.program_sskey = s.program_sskey
AND m_03.iss = s.iss AND m_03.application_sskey = s.application_sskey
AND m_06.iss = s.iss AND m_06.territory_sskey = s.territory_sskey
AND m_02.iss = s.iss AND m_02.product_sskey = s.product_sskey
AND m_05.iss = s.iss AND m_05.customer_sskey = s.customershipto_sskey
AND m_01.iss = s.iss AND m_01.customer_sskey = s.customerbillto_sskey
AND m_07.iss = s.iss AND m_07.warehouse_sskey = s.warehouse_sskey

--#BLOCK_END# MapAll

/*****/
-- Set join order for SQL Server
/*****/

--#BLOCK_BEGIN# ForcePlanOff

SET FORCEPLAN OFF

--#BLOCK_END# ForcePlanOff

```

```

/*****
-- Look for unjoined data, Report on processed rows
*****/

--#BLOCK_BEGIN# SPResults

DECLARE @unjoined INT
DECLARE @processed INT

BEGIN

SELECT @processed = (
SELECT COUNT(1)

FROM OrderStage
)

SELECT @unjoined = (
SELECT @processed - COUNT(1)

FROM OrderStage_MAP
)

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'UNJOINED', @unjoined

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', @processed

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'INSERTED', @processed - @unjoined

END

--#BLOCK_END# SPResults

/*****
-- Index this temp table
*****/

--#BLOCK_BEGIN# IndexMap

EXEC('
CREATE INDEX XOrderStage_MAP ON OrderStage_MAP
(
    iss, ss_key, date_key, ikey
)
')

--#BLOCK_END# IndexMap

--#TEMPLATE_END# map_keys
--#TEMPLATE_BEGIN# upd_unj

/*****
--
-- Copyright * 1997, Epiphany Marketing Software, Inc. All Rights Reserved
--
-- upd_unj
--
-- Epiphany Marketing Software
--
-- Update all dimension keys to 'UNKNOWN' in staging table
-- where referential integrity fails
--
*****/

```

```

/*****
-- Count the number of rows to update in the staging table - that is, those
-- that have at least one Foreign key where referential integrity fails
*****/

--#BLOCK_BEGIN# CountUnj

BEGIN

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'PROCESSED', COUNT(1) FROM OrderStage

INSERT INTO adaptive_template_profile (token_name, number_rows)
SELECT 'MODIFIED', COUNT(1)
FROM
    OrderStage s
WHERE 1=0
OR NOT EXISTS (SELECT 1 FROM ProgramMap_B m_04 WHERE m_04.iss = s.iss AND m_04.program_skey =
program_skey)
OR NOT EXISTS (SELECT 1 FROM ApplicationMap_B m_03 WHERE m_03.iss = s.iss AND
m_03.application_skey = application_skey)
OR NOT EXISTS (SELECT 1 FROM TerritoryMap_B m_06 WHERE m_06.iss = s.iss AND
m_06.territory_skey = territory_skey)
OR NOT EXISTS (SELECT 1 FROM ProductMap_B m_02 WHERE m_02.iss = s.iss AND m_02.product_skey =
product_skey)
OR NOT EXISTS (SELECT 1 FROM CustomerMap_B m_05 WHERE m_05.iss = s.iss AND m_05.customer_skey
= customershipto_skey)
OR NOT EXISTS (SELECT 1 FROM CustomerMap_B m_01 WHERE m_01.iss = s.iss AND m_01.customer_skey
= customerbillo_skey)
OR NOT EXISTS (SELECT 1 FROM WarehouseMap_B m_07 WHERE m_07.iss = s.iss AND
m_07.warehouse_skey = warehouse_skey)

END

--#BLOCK_END# CountUnj

/*****
-- Update foreign keys where referential integrity fails
*****/

--#BLOCK_BEGIN# UpdateUnjprogram_skey

UPDATE OrderStage SET program_skey = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM ProgramMap_B m
WHERE m.iss = OrderStage.iss AND m.program_skey = OrderStage.program_skey)

--#BLOCK_END# UpdateUnjprogram_skey

--#BLOCK_BEGIN# UpdateUnjapplication_skey

UPDATE OrderStage SET application_skey = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM ApplicationMap_B m
WHERE m.iss = OrderStage.iss AND m.application_skey = OrderStage.application_skey)

--#BLOCK_END# UpdateUnjapplication_skey

--#BLOCK_BEGIN# UpdateUnjterritory_skey

UPDATE OrderStage SET territory_skey = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM TerritoryMap_B m
WHERE m.iss = OrderStage.iss AND m.territory_skey = OrderStage.territory_skey)

--#BLOCK_END# UpdateUnjterritory_skey

--#BLOCK_BEGIN# UpdateUnjproduct_skey

UPDATE OrderStage SET product_skey = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM ProductMap_B m
WHERE m.iss = OrderStage.iss AND m.product_skey = OrderStage.product_skey)

--#BLOCK_END# UpdateUnjproduct_skey

```

```

--#BLOCK_BEGIN# UpdateUnjcustomershipto_sskey

UPDATE OrderStage SET customershipto_sskey = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM CustomerMap_B m
WHERE m.iss = OrderStage.iss AND m.customer_sskey = OrderStage.customershipto_sskey)

--#BLOCK_END# UpdateUnjcustomershipto_sskey

--#BLOCK_BEGIN# UpdateUnjcustomerbillto_sskey

UPDATE OrderStage SET customerbillto_sskey = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM CustomerMap_B m
WHERE m.iss = OrderStage.iss AND m.customer_sskey = OrderStage.customerbillto_sskey)

--#BLOCK_END# UpdateUnjcustomerbillto_sskey

--#BLOCK_BEGIN# UpdateUnjwarehouse_sskey

UPDATE OrderStage SET warehouse_sskey = 'UNKNOWN'
WHERE NOT EXISTS (SELECT 1 FROM WarehouseMap_B m
WHERE m.iss = OrderStage.iss AND m.warehouse_sskey = OrderStage.warehouse_sskey)

--#BLOCK_END# UpdateUnjwarehouse_sskey

--#TEMPLATE_END# upd_unj

```

Note, additional semantic types and adaptive templates can be imported into the system 100.

EPIPHANY, INC. **SOFTWARE RELEASE 3.2: CLARITY UPDATE** **NEW PRODUCT: RELEVANCE** **RELEASE NOTES**

Revision 1.16
 August 7, 1998

Introduction

Welcome to Clarity 3.2 and Relevance! This document will summarize the features of Epiphany Software's newest Enterprise Relationship Management software applications. Included are feature enhancements to Epiphany's Clarity 3.1, known items to look out for, and where to go for further information.

Note on migrating from Clarity 3.1 to Clarity and Relevance 3.2: There is a high level of added functionality, tools, and a much-improved streamlined structure of Epiphany's database schema. The migration process is therefore best served by providing personalized service to migrating customers via Epiphany's Professional Services Department. This process is not articulated in this document; migration will instead be realized through hands-on implementation by Epiphany's services in conjunction with the customer's IS / IT department.

Section I: Feature Enhancements

This section summarizes the new features provided in Release 3.2.

New Features for IS teams

□ ***NEW ADAPTIVE SCHEMA GENERATOR***

Release 3.2 employs a more powerful, integrated metadata architecture with the EpiCenter Enterprise Manager, a new and user-friendly GUI (EpiMgr.exe). EpiMgr consolidates and integrates all the variables and parameters that drive the Epiphany system, facilitating the job of the administrator to create, modify, and control the company's metadata.

The Enterprise Manager provides for many newly added functions:

The ability to import and export part or all of your metadata,

Well organized, pop-up forms facilitating metadata creation and configuration, and

Clear displays of vital information regarding dimensions, facts, aggregates, extraction settings, security settings, and even ticksheets.

□ ***EPICHANNEL EXTRACTORS, SEPARATE METADATA AND DATAMART DATABASES***

The 3.2 EpiChannel extractors are defined by the same GUI as is the star schema, which integrates SQL and non-SQL steps into a multi-stage workflow. Extraction statements have a more flexible mapping to the destination table and can use adaptive SQL either for database vendor independence or for adaptive extractions. Separate databases for metadata and aggregate data provides for easier maintenance, less intrusion or damage from mishaps, and easier error recovery.

Extraction set identification is more flexible and controllable. Trial runs and limited return set runs are supported. All runs are monitorable via the NT performance monitor, featured below.

❑ ***CONFIGURABLE, INFORMATIVE EXTRACTION LOGS***

Release 3.2 provides substantially increased and clearer log output from extractions and aggregate builds. Additionally, error handling is much more robust and features optional verbosity levels. These numerous, user-settable verbosity or “debug” levels control the amount of output the user would like to see from the back-end operation. These debug settings also provide for breakpoints, which may be established between or even within SQL statements. Extraction and aggregate build output is printed out in real-time as well as placed in separate logs with distinct and valuable information.

❑ ***NEW SECURITY SYSTEM***

This new feature provides ticksheet access control at multiple levels, from entire ticksheets to line item attributes and resulting facts. User access can be limited to any set of values on any field. Moreover, the Epiphany user and password management can be completely integrated into the NT security system, avoiding the need to maintain duplicate lists of users and groups.

❑ ***SPEED***

Hotswaps, used as safeguards in error recovery for extractions, are now instantaneous due to a resourceful design change in its method.

Query reports can complete from two to many times faster, depending on the query and the environment. Certain corner case queries that ran for a prohibitively long duration in Release 3.1 can now be realized in moments.

❑ ***OPTIONAL PERFORMANCE MONITOR***

Through a standard NT monitoring tool, real-time graphical observation on the progress of an extraction can be easily observed. This tool assures the observer that the extraction is in fact progressing; moreover, it indicates what step the extraction is currently processing.

New Features for Users

❑ ***NEW PRODUCT! RELEVANCE***

This new analytical tool set lets you identify trends, forecast quarter projections, discover what is doing well and what is not, find clusters of buying behavior, spot patterns in spending, and capture a graphical bird's-eye view of your whole business.

❑ ***STREAMLINED BROWSER USER INTERFACE***

The user interface has been streamlined to better handle many filters and options. These are now set in pop-up windows, thus simplifying the main page and making it easier for the user to see all the aspects of ticksheet selections simultaneously. The consolidation of filter settings and newly expanded display options on one summary screen will be appreciated by both new users and experienced creators of complex reports.

❑ ***TICKSHEET BUILDER ADDITIONS***

New Ticksheet Builder provides clearer ticksheet building and measure mapping. Also added is the provision for cloning one type of ticksheet to another, reducing the need for repeated data entry and providing for better homogeneity among ticksheets.

❑ **REPORT GALLERY**

Queries are now easy to save and restore with a rich full-page Report Gallery that lists saved reports with descriptions, save dates, creator and permissions. A newly implemented security provision also provides for easy user, group and global permission settings for reads and modifies on all saved query reports. This permits users to seamlessly share reports with others while keeping sensitive data secure. The Gallery can also list non-Epiphany documents or sites on the web, giving the user one location for reports regardless of their source or type -- Epiphany reports, spreadsheets, text documents, external databases, published reports, whatever.

❑ **NEW JAVA-BASED QUERY PROCESSOR – FASTER QUERY RESPONSE**

The 3.2 release utilizes a new Java-based query architecture that is faster and more robust than that of the 3.1 release. The Epiphany system still uses plain web browsers without the need for Java, ActiveX or Plug-ins on the clients. On the server, however, Epiphany now uses an advanced multi-threaded query processor written entirely in server-side Java. This new machinery makes a direct ISAPI connection to the IIS web server, resulting in a system of superior performance than that of 3.1's CGI-based architecture.

Section II: Known Issues

1. Installing the Application Server over a pre-existing installation.

When installing and overwriting the same instance of a previous Epiphany Application Server installation, especially if the previous installation's appserver has been or is still running, there might be trouble overwriting some files because of a sharing violation.

Resolution: Although the installation script stops and restarts the WWW service (W3SVC) for you, perhaps something precluded this operation. Stop W3SVC, delete the file in question, rerun the installation, then restart the WWW Service.

If the Application Server was running with the same instance name and parameters, the installation will abort upon encountering a sharing violation. Take care to restart the WWW service by hand in case the install script aborted in the middle and stopped the service without recovering it.

2. Uninstalling Epiphany Software

Currently, uninstalling the Epiphany Application through "Add/Remove Programs" under Windows is unreliable. In the interim, to uninstall the application, remove the instance directory in question or the entire Epiphany directory. It is not necessary to delete the corresponding instance key(s) in the registry; in any case, the path is [HKEY_LOCAL_MACHINE\SOFTWARE\Epiphany], and the sub tree is [HKEY_LOCAL_MACHINE\SOFTWARE\Epiphany\Instances\<instance name>].

3. SQL Server Version

Currently, Epiphany Software functionality is only supported when running with SQL Server Version 6.5 and SVP Pack 3 or later. SQL Server Version 7.0 is not compatible with Clarity or Relevance Version 3.2.

Support for 7.0 is forthcoming; in the meantime, the Epiphany installation procedure does not preempt installation on a system that is running SQL Version 7.0. It is up to the installer to check for this before installing the Epiphany Application Server option.

4. Internet Information Server Version

Release 3.2 is supported with IIS Version 3.0; Release 3.3 will be supported with IIS Version 4.0.

5. EpiMgr.exe: Importing Exported MetaData.

This note is applicable in the case where the metadata was exported in sections to facilitate importing metadata, also in sections. This is a useful alternative to importing a large metadata database all at once.

When importing metadata, particularly to a newly built database, there are likely some dependencies which one set of metadata can have on another set within the metadata's hierarchical structure. Examples of "sets" of metadata are: Constellations, Extractors, etc. Dependencies can lie within these sets, such as "saved queries"; these require a reference to a corresponding ticksheet type as well as security entries which correspond to the read, write and default permissions contained within a saved query.

Also when re-importing one aspect of the metadata into a populated database, links to other metadata may be lost if this order is not adhered to. For example, re-importing measures into existing metadata will sever all the measure mapping links to the ticksheets. Then re-importing the ticksheets will re-link the mappings which will cause a loss of the saved queries.

So, a rule of thumb is to follow the order listed below when importing metadata into to a new meta-database, presuming the exported was in fact exported in sections:

- a. Constellations
- b. Extractors
- c. Measures
- d. Ticksheets
- e. Security
- f. Saved Queries

6. Internet Explorer 3.0x on Windows NT.

If one user installs an IE 3.0x browser, then a different user runs Clarity using this browser, the first element of every listbox will be selected on the ticksheet in place of the default selection. This occurs only on Windows NT; no workaround is available at this time.

7. Epiphany Results Windows: IE3.0x

On a PC running Epiphany queries on IE 3.0x, running the very first query results in a new IE window creation which will pop up in front of the ticksheet window which launched the query.

After the first query, if the results window remains on the desktop and is sitting behind any other windows, subsequent query results will appear in that same window and will not pop forward.

The indication of a query's progress appears in the status bar of the results window and not the ticksheet window. When the results window does not pop forward, the user may get the impression that the query either did not become invoked or ended prematurely, when this is in fact not the case.

On a Macintosh, this same behavior occurs with either Internet Explorer or Netscape browsers, adding that even the first results window created may not pop-up in front of the ticksheet window.

8. Browsers, Epiphany Software, and RAM

HTML tables may consume large amounts of system memory. Therefore, when running reports on computers with 16 → 48 MB of RAM, the system may warn that it is running low on memory. This is especially true with particularly long queries, such as those listing a large number of rows and columns, or those where many measures have been requested. This has been seen on Macintosh systems with 16 MB of RAM, running Internet Explorer 4.0x.

The user may also find that downloading spreadsheets will take a particularly long time, specifically with queries that produce thousands of rows. This has been seen on Windows 95 systems running 32 MB RAM and Netscape 3.0x. This is currently being investigated; the results will be addressed and updated in a coming release.

Therefore, although a system with 16 → 48 MB of RAM can certainly run Clarity and / or Relevance, 64 MB of memory is recommended as a minimum for a client system running Clarity and/or Relevance for smoother operation.

The Workaround provided for this and other long query scenarios is the provision for reporting the results directly into spreadsheet format. This option is useful in any configuration when running queries with a large number of rows, which typically takes much longer to display in a browser than in a spreadsheet.

9. Corel Draw, Other Software Packages and IE4

It has been discovered that some applications, when installed on a system after IE4 has already been installed, write over some files that IE4 uses for correct Java Script functionality. CorelDraw is one example of a software product in this category.

IE4 will appear to work fine with web sites and applications that do not use these files. Other web sites and applications which use certain Java operations, Epiphany Clarity and Relevance Query Machinery being among them, will be severely disabled.

We have yet to discover which files are affected. Today, the only workaround is to re-install IE4 once this scenario is encountered. It is yet unknown what effect this will have on the functionality of the offending applications; upon more investigation and customer feedback, the answers will become clearer.

10. Java Script Versions

On some installations of Internet Explorer 3.0x, Java Script Version 1.0 is provided. Epiphany does make the claim that Java Script 1.1 is required, although we have seen that queries can in fact be invoked successfully with Java Script Version 1.0.

If JavaScript Version 1.0 is installed, the Login and Home screens will display a message saying that you do not have 1.1 and therefore cannot proceed. Nevertheless, it is likely that you can in fact proceed with your queries.

If this message is particularly annoying and you are having no trouble with queries, this message can be eliminated in the templates directory of the Epiphany software; the files involved are "login.template" and "toplevel.template".

11. Ticksheet Displays: Netscape 3.0x

In some environments, running Epiphany ticksheets on Netscape Versions 3.0x may display an alternate colored background, somewhat pinkish and textured instead of the standard off-white background. This may happen when resizing the window, when displaying another window above the ticksheet window, etc. This peculiarity has been exhibited with NS3.0x exclusively.

12. Netscape 4.0x

In Netscape 4.0x only, we have seen the browser crash when quickly clicking on different Ticksheet type and Dataset hyperlinks in the left-hand menu list. This can occur when, repetitively, each click to a new ticksheet hyperlink is made before the previous ticksheet has a chance to render.

13. Web Builder - Measures

In the Measure Builder Window of the Web Builder utility, there is a drop down list under Measure Terms in the Fact Column/Dimension Table field. This list includes all the dimensions that were defined in EpiMgr for the current constellation. The dimensions Date and Transtype, which are built by default with every metadata creation, are invalid for this field and incorrectly included in this drop down list.

Choosing either of these base dimensions as a measure element then saving the configuration as such will cause Web Builder to post an exception during the save, even though the save does complete what it can. However, the Application Server will not launch until the Web Builder measure described above is re-modified to exclude the base dimensions.

A second circumstance with Web Builder's measure window is the allowance of multiple instances of them to be opened. If two measure windows are opened, one is closed, and a ticksheet form is opened

thereafter, Web Builder may exit with an error. Open only one measure builder window within a Web Builder session.

Lastly: Launch Web Builder. Open the Measure Builder Window only. Make changes only to the measure terms of one measure. Close the Measure Builder window, then close Web Builder. Web Builder will exit without prompting for "saving changes before exiting", and the measure term changes will be lost. Changes of any other type will prompt the user to save/not save the changes upon exit.

14. Charts in Relevance

In Release 3.2 when running Relevance Ticksheets of type "Trends" or "Quarter Projections," only line charts are currently displayed regardless of choosing 2D or 3D charts in the display options window.

On Best/Worst type ticksheets, no charts are available. This is by design; the reason that charts are still an option is because of drill-down and save requests that may sequentially follow, for which charts can be a relevant choice.

15. Installation and the MSSQL Server

To run the Application Server or the Remote Administration tool package, the SQL Client utilities must be loaded onto the system. SQL may be installed after the Epiphany installation has been completed, although it must be there at run time.

The installation program does not perform a check for the presence of the SQL Server. The Epiphany installation will complete successfully, and the absence of SQL will be discovered at run time by the lack of proper functionality. Therefore, make sure that the correct version of the SQL Server is in fact properly installed (see Item 3 of this Section).

16. Downloading Spreadsheets

There are three minor issues here, all which will be resolved in the coming release:

- 1) The download spreadsheet option will not display within the results browser window when using Internet Explorer 3.0x as it does in IE4.0x. Instead, the full path and name of the target sybk file appears in the results window. A simple workaround is to highlight and capture the full path and name including the http prefix from the results window and paste it into the browser's address field, thereby displaying the spreadsheet in the window as intended.
- 2) If a precision of one or two decimal places is set in the display options, the spreadsheet display will not follow this setting as the HTML display does. Instead, no decimal places are displayed. The workaround for this is to select the data cells and modify the number format by hand.
- 3) If non-additive measures are chosen for a report, the Epiphany software as designed will not display data for percentage columns in the HTML report even if this display option is chosen. When spreadsheets are displayed however, the percentages do appear when they shouldn't. The workaround is to delete the percentage columns manually.

17. Duplicate Log Entry Errors

A SQL Server bug may cause identity values to be duplicated in Epiphany query logs. In the run-time AppServer logs provided, an error may appear as follows:

```
java.sql.SQLException: [Microsoft][ODBC SQL Server Driver][SQL Server]Violation
of PRIMARY KEY constraint 'PK_query_log_filter_7088BE1D': Attempt to insert
duplicate key in object 'query_log_filter'.
```

This is NOT a serious error and will in no way result in corruption of metadata. In any case, the remedy is as follows: Run the SQL Server tool (ISQL/w) and open the file "metacheck.sql", provided in the Win32 directory of your Epiphany AppServer installation. Run this SQL command list against the metadata database, and the errors should clear up.

18. Windows 95

Before installing Release 3.2's remote administration utility on a Windows 95 system, we recommend keeping a backup of the ".DLL" and ".OCX" files that reside under the \Windows\System directory, namely those of the same name as the DLLs and OCXs that come under the i386\Dll directory of Epiphany's installation kit, listed below. Epiphany does overwrite these DLLs, and although all should be compatible versions, a DLL may be overwritten with a version which another application will not tolerate. We'll be keeping an eye on this to expose any known incompatibilities

The DLLs to be aware of are as follows:

<i>adme.dll</i>	<i>gsgif32.dll</i>	<i>msrd2x35.dll</i>	<i>snjawt11.dll</i>
<i>asycfilt.dll</i>	<i>gsjpg32.dll</i>	<i>msrdo20.dll</i>	<i>snjnet11.dll</i>
<i>comcat.dll</i>	<i>gsprop32.dll</i>	<i>msrepl35.dll</i>	<i>snjreg.exe</i>
<i>comctl32.ocx</i>	<i>gsw32.exe</i>	<i>msvbvm50.dll</i>	<i>snjrt11.dll</i>
<i>comctl32.ocx</i>	<i>gswag32.dll</i>	<i>msvcrt40.dll</i>	<i>stdole2.tlb</i>
<i>comdlg32.ocx</i>	<i>gswdll32.dll</i>	<i>odbcji32.dll</i>	<i>tabctl32.ocx</i>
<i>ctl3d32.dll</i>	<i>msdtcprx.dll</i>	<i>odbcjt32.dll</i>	<i>vbajet32.dll</i>
<i>dao350.dll</i>	<i>msjet35.dll</i>	<i>odbctl32.dll</i>	<i>xolehlp.dll</i>
<i>dtccm.dll</i>	<i>msjint35.dll</i>	<i>oleaut32.dll</i>	
<i>dtctrace.dll</i>	<i>msjter35.dll</i>	<i>olepro32.dll</i>	
<i>dtcutil.dll</i>	<i>msmask32.ocx</i>	<i>rdocurs.dll</i>	

19. Security - Users and Multiple Groups

When a user is a member of multiple groups, the access rights associated with each group will be handled in an additive fashion. The user and administrator must be aware that the security logic handles these additive rights on a column by column basis rather than on a group by group basis, which will be changed in the next release.

The implications of the current method is that when a user is a member of multiple groups which have conflicting, overlapping, or exclusive dimension level access permissions set, it may result in an undesired security breach.

Example:

Group A has permission to see the sales figures for Product 1 in the Western Region.

Group B has permission to see the sales figures for Product 2 in the Eastern Region.

If User X is a member of both groups A and B, he/she will be able to see the figures for Product 1 and Product 2 in *both* regions.

20. Database Entries beginning with spaces

If there are data values that begin with spaces in the source database system, these leading spaces will be truncated when displayed as attributes in the browser query results. The same is also true when these values are entered as filters. Because the spaces do exist in the SQL database, filters on this type of data will fail.

21. Web Builder – Query Fills

If the SQL Server is stopped and started during a Web Builder session is stopped and started, the "Fill from Query" utility used in defining filter groups will result in the message: "No records returned from the query." From then on, even if the SQL server is restarted, a "Fill from Query" operation will not operate and will not return a result. Web Builder must be stopped and restarted to reactivate this utility.

22. Installing while EpiAppServer is running

If an installation is launched with the same instance name as an Application Server that is running, the installation will fail. A dialog box will say that "Error 115 occurred in copying the files" and that EpiServerIni.dll failed to self-register. The installation will then abort. To reinstate the server, make sure

the WWW (W3SVC) service is restarted, and the EpiAppServer is stopped and restarted. To re-install, stop the EpiAppServer, perform the installation, then restart the EpiAppServer.

23. Defining Filter Blocks

Do not use valid HTML code character sequences as labels for filter blocks; they will not display correctly in the results windows.

24. Clientlogo

Any customized logo to be displayed in a ticksheet must be named "clientlogo.gif" and placed in the Web/WWWRoot/images directory.

25. Current DataMart Settings

This item is just a reminder regarding running back room operations, (extractions (extract.exe) and aggregate builds (agg.exe)). The "current" datamart letter (A or B) set in Epimgr's configuration will and should be set to the opposite of what the real active datamart will be when running queries. After running an extraction, a hotswap, provided as a pre-built job step, switches the active letter, so queries will be run against the latest extracted data.

SECTION III: DOCUMENTATION REFERENCES

1. Installation

For Installation instructions, system requirements, listings and installation of required supporting software packages, refer to the Epiphany System Guide for Clarity and Relevance 3.2, Appendix A.

2. Tools

For EpiMgr and Web Builder Tools, refer to the Epiphany System Guide for Clarity and Relevance 3.2, Chapters 3 and 4 respectively.

3. Extraction, Aggregate Builds (Back Room Processing)

Refer to the Epiphany System Guide for Clarity and Relevance 3.2, Chapter 2.

4. Optional Performance Monitor

An optional performance monitor setup is included with your Epiphany software. This provides the ability to track the progress of an extraction job using WinNT's charting tool. Installation and operation of this handy tool comprises the last section of the Epiphany System Guide for Clarity and Relevance 3.2, Appendix A.

5. Application Server

AppServer operation, security, refresh, logs, etc are described in the Epiphany System Guide for Clarity and Relevance 3.2, Chapter 5.

6. Web Browser Queries

Ticksheet usage is very straightforward. Ticksheets bear extensive on-line help links which will facilitate maneuvering through them. A brief summary and picture of the ticksheet layout opens Chapter 4 of the Epiphany System Guide.

Epiphany System Guide



Clarity and Relevance 3.2

July 1998

EPIPHANY CONFIDENTIAL

09625518.072500

Copyright and Trademarks

Copyright © 1998 by Epiphany, Inc. All rights reserved.

Epiphany System Guide: Clarity and Relevance 3.2

This document and the software described in it is furnished under license and may be used or copied only in accordance with such license. Except as permitted by such license, the contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, without the prior written permission of Epiphany, Inc.

This document contains propriety and confidential information of Epiphany, Inc. The contents of this document are for informational use only, and the contents are subject to change without notice. Epiphany, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this book.

Unpublished rights reserved under the copyright Laws of the United States.

Clarity™ and Relevance™ are trademarks of Epiphany, Inc. All other products or name brands are trademarks of their respective holders.

Printed in the USA

Part Number DOC-CL-3-2-001

EPIPHANY CONFIDENTIAL

09625518.072500

CONTENTS

Chapter 1 Introduction	ix
About This Guide	ix
How To Use This Guide	xi
Chapter 2 Basic Concepts	1
Epiphany Database Schema	1
The EpiCenter	3
Dimension Tables	5
Dimension Roles	5
Date and Transtype Dimensions	6
Attributes	6
Fact Tables	7
Epiphany System Overview	8
Chapter 3 Epiphany Database Extraction	11
Basic Concept: The Epiphany System Is Metadata	13
Extraction Phases: An Overview	15
Load Phase	15
Data Merging	16
Aggregate Building	16
Data Stores	16
The Role of Jobs in Extraction	18

00625518.072500

Extractors	20
SQL Statement Extraction Step	22
SQL Macros	23
Staging Tables	23
External Tables	24
Semantic Instance Extraction Step	25
Applying a Semantic Type	26
System Calls	26
Aggregate Building	28
The Aggbuilder Program	31
The Aggregate Building Process	32
Custom Fact Indexing	33
The UNKNOWN Dimension Row	34
Referring to the UNKNOWN Row during Extraction	35
The Update Unjoined Semantic Type	36
Normal Extraction Order	36
Running Jobs: EpiChannel	37
The EpiChannel Command Line	38
EpiChannel Registry Keys	40
Output Files	41
Extracting New Rows Only	41
How EpiChannel Identifies Data To Be Extracted	42
EpiChannel Debugging	44
Job Output	44
Error Messages	50
EpiChannel Debugging Levels	50
Setting Breakpoints	51
Trial Runs	52
EpiChannel Output	52
Log and Working Directories	53
Log Files versus Log Databases	53
Monitoring Jobs	53
Running the Performance Monitor	54
Mirroring: A and B Tables	55

09625518.072500

Chapter 4 EpiCenter Manager	57
Planning Your EpiCenters	57
Getting Started	59
Basic Concept: The Uniform Treatment of Time	65
Working with an Existing EpiCenter	66
The EpiCenter Manager Window	67
Working with EpiCenter Manager	67
Setting Up an EpiCenter	69
Configuration	69
Base Dimension Tables	69
Defining Dimension Aggregates	71
Dimension Aggregate Browsing	74
Setting Up a Constellation	75
Defining Dimension Roles	76
Degenerate Dimensions	77
Defining Fact Tables	77
Defining Fact Columns	78
Custom Indexes	79
Fact Aggregate Browsing	80
Aggregation and Aggregate Grouping	81
Measures and Ticksheets	86
Basic Concept: Uniform Transactional Data	87
Extraction	90
Data Stores	90
The Data Store Dialog Box	90
Modifying the Default Data Stores	93
External Tables	94
Extractors	95
The Extractor Steps Dialog Box	96
Jobs	102
Adding Extractors and System Calls as Job Steps	105
Configuring E-Mail	106
Truncating Tables	107
Purging EpiMart Tables	108

Security	109
Dimension Column Access	111
Defining Groups	112
Global Queries	112
Synchronized Groups	113
Defining Users	115
Basic Concept: Adaptive Architecture	117
Generating Schema	121
Exporting/Importing Metadata	122
Chapter 4 Web Builder and Ticksheets	125
Generating a Report	125
Attributes and Measures	126
Display Options	127
Filters	128
Web Builder Overview	129
Getting Started	130
The Web Builder Window	130
The Main Menu	131
Right-Click for Pop-up Menus	133
Web Builder Icons	133
Setting System Properties	134
Defining Datasets	135
Defining Measures	136
Defining Measure Terms	137
Reverse Polish Notation	140
Defining Dictionary Entries	142
Creating a New Ticksheet	143
Ticksheet Types	145
The Ticksheet Dialog Box	146
Defining Attributes	147
Defining Column Elements	148
Measure Mapping	149

Defining Filters	152
Defining Filter Blocks	152
Defining Filter Groups	154
Configuring Relevance Ticksheets	156
Chapter 5 Epiphany Application Server	159
Starting and Stopping the Server	162
Running as a Service	162
Determining if the Application Server Is Running	162
Running as a Console Application	164
For Authentication to Work	165
Command-line Arguments	167
Refreshing the Application Server	167
The Refresh Command Line	169
Invoking RefreshApp	171
The EpiAppService Program	172
Command Syntax	173
EppiAppService Command Examples	175
The Application Server's Registry Keys	175
The Epiphany Proxy	178
Proxy Logging	179
Application Server Logging	181
Log File Location	182
Log File Naming Conventions	182
The Server Log	183
The Security Manager	184
Save and Restore Manager	185
Clarity and Relevance Applications	185
Application Server Security	186
Authentication Modules	188
Authentication Module Tips	189
Configuring Output Templates	192
Environment Variables	195
Customizing Error Messages	197
Name Replacement	198

Appendix A Installation	201
Epiphany System Requirements	201
Installation Overview	203
Installing Windows NT Server 4.0	204
Windows NT Setup	204
Windows NT Networking	204
Microsoft Internet Information Server (IIS) Setup	206
Install the Service Pack	207
Install Microsoft Office	207
Installing Microsoft SQLServer	208
Install the Service Pack	210
Microsoft SQLServer Setup	210
Setting Up Your Databases	210
SQLServer Configuration	212
Installing the Epiphany Software	213
Application Server	215
Remote Administration	216
Setting Up the Epiphany Application Server	216
IIS 3.0 Settings	217
IIS 4.0 Settings	219
Manual Chart Install	220
Setting Up NT Performance Monitoring for Extraction (Optional)	222
After Installing the Epiphany Software	224
Appendix B Epiphany Macros	225
System Call Macros	225
System Call Macro Syntax	225
Epiphany SQL Macros	229
Vendor-independent Macros	229
SQL Macro Usage	230
SQL Macro Notes	231

Appendix C EpiCenter Configuration	243
Configuration	243
Transaction Types	246
Measure Units	247
Appendix D Date Dimension Fields	249
Appendix E Physical Type Values	255
Appendix F Writing Staging SQL Statements	259
Base Dimension Staging SQL Statements	259
Duplicate sskey's	262
Dimension Staging Queries with Joins	262
Constructing Base Dimension Queries with DISTINCT Fact Values	263
Fact Staging SQL Statements	263
Using External Tables as Inputs to Staging Queries	266
Appendix G Semantic Types	267
Dimension Semantic Types	267
Slowly Changing Dimensions	267
Latest Dimension Value	269
First Dimension Value	270
Initial Dimension Load	271
Fact Semantic Types	272
Update Unjoined (Optional)	272
Custom Fact Index	273
Transactional	273
Transactional/State-like	274
Transactional/State-like/Force Close	275
Transactional/Inventory	277
Transactional/Inventory/Force-zero	277
Pipelined	277

Appendix H Export/Import of Metadata	279
Metadata Overview	279
Replacing Existing Metadata on Import	281
Actuals and Agg Metadata	282
Export File Format	282
Appendix I Troubleshooting	285
Registry Editor Warning	285
SQLServer Error Message	286
Cannot Connect to the Server	286
Application Server Error Messages	286
User Cannot Log In	287
Additional Action to Take If User Still Cannot Log In	289
Allow Interaction with Desktop Problem	290
Out of Memory	290
VirtualMartDatabase Key Missing Error	291
Invalid Object DATE_O Error	291
Not a Valid Application Error	292
Internal Windows NT Error	292
EpiQuery Engine Database Connection Open Failure Exception	293
Charts Do Not Display	293
GIF Images Fail to Display on Web Pages	297
No Results Available for a Query	298
Result Page Error: Extraction Date Unknown	298
Web Server Message: Object Not Found	299
Browser Crashes When Retrieving Results from Application Server	301
Refresh Program Fails	301
Application Log Full Error	301
Application Server Log Security Problem	302
Glossary	303

INTRODUCTION

Welcome to Clarity and Relevance 3.2, Epiphany's Enterprise Relationship Management (ERM) suite of decision-making tools that give you quick access to detailed data in simple, straightforward terms.

Clarity provides a window into the data that exists throughout the enterprise—leads, customers, orders, production, and financial reporting. Using Clarity reports and charts, you can analyze complex data, focusing on the aspects that interest you.

Relevance helps you to quickly understand your current data and enables you to forecast quarterly projections and future trends. Using a Web browser, can view all of this data numerically or graphically.

About This Guide

The *Epiphany System Guide* is intended for database administrators and consultants who will install and configure the Clarity and Relevance 3.2 software and will be responsible for maintaining an Epiphany site.

This *Guide* consists of the following chapters and appendices:

Chapter 1, *Basic Concepts*, describes the Epiphany database schema and introduces Epiphany-related data warehousing concepts. Additional basic concepts are presented as special topics in Chapters 2 and 3.

Chapter 2, *Epiphany Database Extraction*, describes the Epiphany database extraction process.

Chapter 3, *EpiCenter Manager*, explains how to set up your organization's data warehouse, which is called an EpiCenter.

Chapter 4, *Web Builder and Ticksheets*, is a guide to Web Builder that explains how to create and modify *ticksheets*. At the front end, users open a ticksheet in a Web browser and select options for the kind of data they want to display. (A *ticksheet* is form that allows end users to construct queries; it is called a ticksheet because users select, or tick, items on a page.)

Chapter 5, *The Epiphany Application Server*, is an operator's guide to the component of the Epiphany ERM application suite that processes all user requests and returns query data in HTML format.

Appendix A, *Installation*, describes the Epiphany system prerequisites and provides instructions for installing and configuring Windows NT 4.0, Microsoft SQLServer, and the Epiphany Clarity and Relevance 3.2 software. Instructions for setting up your Epiphany Application Server are also given.

Appendix B, *Epiphany Macros*, defines the Epiphany-supplied system calls and SQL macros.

Appendix C, *EpiCenter Configuration*, describes the configuration data for a default EpiCenter.

Appendix D, *Date Dimension Fields*, defines the date dimension fields used by the Epiphany system.

Appendix E, *Physical Type Values*, defines the database type translations for the physical types used by the Epiphany system.

Appendix F, *Writing Staging SQL Statements*, explains how to write these SQL statements for base dimension tables and fact tables. Instructions for using external tables as inputs to staging queries are also given.

Appendix G, *Semantic Types*, describes the dimension and fact semantic types.

Appendix H, *Export/Import of Metadata*, presents an overview of the Epiphany system's use of metadata as a basis for a discussion of how the Epiphany export/import metadata feature works.

Appendix I, *Troubleshooting*, describes the Epiphany error conditions and error messages and suggests corrective action.

The *Glossary* defines the terms used throughout this Guide.

How To Use This Guide

A suggested approach to using this *Guide* follows:

- See Appendix A, *Installation*, for instructions on installing and configuring the software and setting up your system.
- Read Chapter 1, *Basic Concepts*, and Chapter 2, *Epiphany Database Extraction*, for the background material you need to set up an EpiCenter. Refer to the *Glossary* for definitions of terms.
- Follow the instructions in Chapter 3, *EpiCenter Manager*, to configure your EpiCenter.
- Run the EpiChannel program, as explained in Chapter 2, *Epiphany Database Extraction*, to extract data from your source systems and place it into the Epiphany tables.
- Start the Application Server as described in Chapter 5, *Epiphany Application Server*, and verify that it is working.
- Construct ticksheets for your organization as described in Chapter 4, *Web Builder and Ticksheets*.
- Refer to the appendices for more detailed information as directed throughout this *Guide*.

0925518.072500

CHAPTER ONE

Basic Concepts

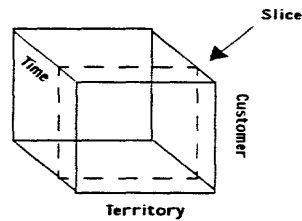
This chapter introduces the Epiphany database schema and provides an overview of the Epiphany system.

Epiphany Database Schema

The Epiphany database schema is based on the dimensional data warehouse model. A data warehouse transforms the raw data from an organization's source system databases into data accessible for query and analysis. The data conforms to the organization's business model and is consistent, reusable, and flexible (that is, the data may be re-sorted by any measure the business uses). A data warehouse also provides the tools needed to query, analyze, and publish this data.¹

The dimensional data warehouse organizes data in what may be visualized as a cube. One can figuratively slice along any dimension of the cube of data to obtain information about the intersection of the dimensions at that point. For example, if the cube has the dimensions Customer, Territory, and Time, and one selects order amount as the measure, one could slice through the cube on the Time dimension to determine the total order amount by customer or territory at a specific date.

¹ Ralph Kimball. *The Data Warehouse Toolkit*. John Wiley & Sons, Inc. 1996.
[This is an excellent book about data warehousing.]



In the Epiphany system, an organization's data warehouse is known as an *EpiCenter*. (An organization may have multiple EpiCenters.) The Epiphany Application Suite consists of "front-end" Web-based applications, such as Clarity™ and Relevance™, which are designed for database query and analysis. These applications allow users to query the EpiCenter by selecting dimensions and facts they want to know more about. The results of these queries (shown in reports, charts, and graphs within the user's browser window) are derived from the intersection of these dimensions.

The EpiCenter represents a dimensional data warehouse with database tables organized in a *star schema* (see Figure 1-1, page 3). At the center of a standard star schema is a *fact table* that contains measure data. Radiating outward from the fact table like the points of a star are multiple dimension tables. *Dimension tables* contain attribute data, such as the names of customers and territories. The fact table is connected, or joined, to each of the dimension tables, but the dimension tables are connected only to the fact table. This schema differs from that of many conventional relational databases where many tables are inter-joined.

An advantage of the Epiphany star schema is that it allows an organization's EpiCenter to be regenerated when its business model changes without the need to replace the data warehouse. The Epiphany star schema also facilitates the creation of new EpiCenters for an organization (data from existing EpiCenters can be imported).

The Epiphany star schema subsumes the standard star schema into a larger hierarchical organization (known as a *constellation*) that allows for the sharing of dimension tables by a set of similar fact tables. A constellation is a grouping mechanism for like-structured fact tables.

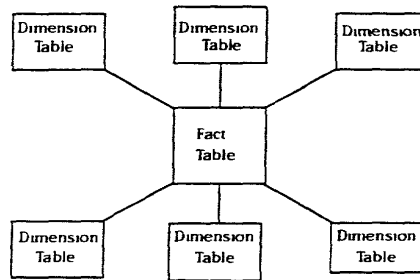


Figure 1-1 Standard Star Schema

As shown in the block diagram in Figure 1-2, page 4, the EpiCenter is the top-level organizing principle. Each Epiphany site has at least one EpiCenter. An EpiCenter is organized into one or more constellations, and a constellation consists of a set of similar facts. Dimension tables are shared by multiple constellations within the EpiCenter.

The EpiCenter

An EpiCenter is comprised of EpiMeta and EpiMart. EpiMeta refers to all of the Epiphany system's metadata tables. (*Metadata* is information about data, not data itself.) EpiMeta defines the schema for the EpiMart tables that will contain the actual extracted, organized data, such as customer, product, and order data. An EpiCenter is an EpiMeta database with its associated internal link to an EpiMart.

The EpiMart consists of fact, dimension, and staging tables. The fact and dimension tables contain actual customer data. Staging tables, which are discussed in Chapter 2, are the first entry point of raw data from the source systems into the EpiMart—an interim stop before it reaches the EpiMart's tables.

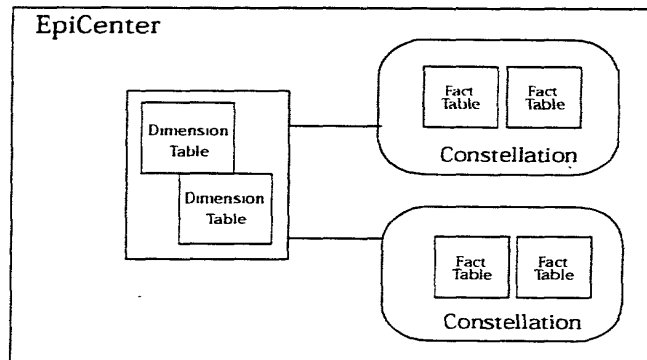


Figure 1-2 The Epiphany Star Schema

To facilitate the building of EpiCenters, Epiphany provides the EpiCenter Enterprise Manager, also called EpiCenter Manager. This is a Microsoft Windows application with an Explorer-like hierarchical structure (see Figure 1-3). The person who designs an EpiCenter (or EpiCenters) for a site uses the EpiCenter Manager's graphical user interface to define the schema for the EpiMart tables.

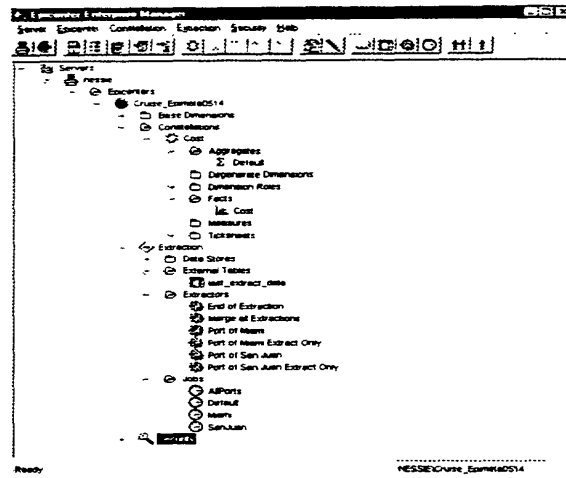


Figure 1-3 EpiCenter Enterprise Manager Window

Dimension Tables

A *base dimension table* consists of dimension columns that hold the actual attributes extracted from the source system. The Epiphany schema allows dimensions to be shared by all of the constellations within an EpiCenter. This is accomplished through the use of base dimension tables that serve as master tables for the entire EpiCenter, and dimension tables within constellations that are assigned “roles,” or aliases. The latter are called dimension roles.

Dimension Roles

A row in a fact table may have several foreign keys that point to the same base dimension table, but the role usage differs. (A *foreign key* is a reference from one table to another table.) For example, a dimension role within an Order constellation and a dimension role within a Customer Support constellation

Basic Concepts

may correspond to the same underlying Master Product List table for data. Within the constellation, these dimension roles may be assigned the role of Product Order List or Customer Support Product List. Multiple roles within a single constellation can refer to the same base dimension.

Dimension roles are defined within a constellation, and all fact tables in that constellation inherit the dimension role. Dimension roles within constellations point to their associated base dimension table for their data. (A dimension role always has an associated base dimension table.)

Date and Transtype Dimensions

The Date dimension and Transtype (transaction type) dimension are assumed common to all constellations, and therefore are automatically provided as base dimension tables. To ensure consistent treatment of time, Epiphany uses a single date dimension throughout the system.

Transaction type dimensions are necessary because the EpiCenter must be able to distinguish among different rows in a single fact table, such as shipping transactions versus bookings, and bookings versus booked returns.

Attributes

Attributes describe a dimension. Some attributes, such as Customer Name, are free-text descriptions, but most have a discrete set of values. For example, a Territory dimension table may consist of its territory key and text that describes each of the organization's sales regions; such as, Eastern, Western, and Southern. It may also contain attributes for a region's subdivisions, such as City and State. Therefore, a hierarchical relationship exists in the dimension table in which the City attribute rolls up (aggregates) into State, which then rolls up into Region.

Region	State	City
Western	CA	Los Angeles
Western	WA	Seattle

Note: In the EpiMart, facts are quantifiable measurements, usually numbers. Attributes of dimensions are usually character strings.

Fact Tables

Fact tables are physical database tables that contain dimension role foreign keys and fact columns. A fact column is a single column in the fact table whose values are numeric, such as *net_price*. In the EpiCenter, a measure is an arithmetic combination of fact columns.

A row in a fact table represents a relationship among a set of values (each value is a separate field in the table). As shown in Figure 1-4, a row may exist for a specific customer number, product number, territory number, and date—all foreign keys that point to base dimension tables for their data—and fact columns, such as the number of units and the price per unit. The table may have millions of rows of data.

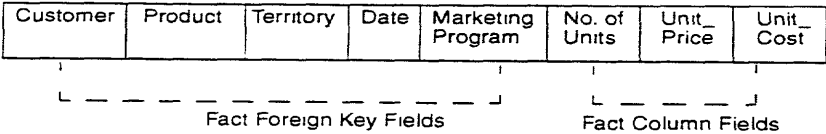


Figure 1-4 Sample Row in a Fact Table

For example, a fact table for Orders might have rows of data that contain fact columns and foreign keys that reference base dimensions, such as Business Unit and Territory. There may also be foreign keys for Customer_Bill-to and Customer_Ship-to that point to the Customer base dimension table. The dimension role metadata defines these foreign keys. (A fact table may have as many foreign keys to a single base dimension table as the business model requires.)

Each of the fact columns of a fact table can be totaled for any dimension and presented in report and chart format.

09625518 072500

The Epiphany database organization allows users to make flexible queries. For example, a Clarity user can query the EpiMart for the names of customers who attended a Boston '96 trade show to determine the dollar amount of orders these attendees purchased as a result of this show.

Epiphany System Overview

An Epiphany site's source data may reside in any number of database management source systems, both relational and non-relational. Examples of supported relational database management source systems (RDBMS) and their related software are—

- telemarketing applications (ACT! and Aurum)
- sales force automation (Siebel and Onyx)
- enterprise resource planning (Oracle Financials, PeopleSoft, SAP R/3)
- customer support applications (Clarify and Vantive)

As shown in Figure 1-5, at the back end of the Epiphany system, data flows from the source databases into the RDBMS server database where the Epiphany database, called the EpiMart, resides. The Epiphany system accesses the source data in a read-only fashion; no changes to an existing source system are necessary. Database updating occurs as part of the database extraction process, which is usually run on a nightly basis.

At the front end, users open a ticksheet in a Web browser on any computer system that supports a JavaScript-enabled browser and select options for the kind of data they want to display. (A ticksheet is user-interface form in Epiphany that allows end users to construct queries; it is called a *ticksheet* because users select, or tick, items on a page.) The Web browser communicates with the Web server, which runs a Java program that queries the RDBMS server for the relevant information. The report is quickly generated and presented as an HTML document. The Epiphany Application Server is the Windows NT Service that connects with a Web server and delivers Epiphany ticksheets and reports. In the Epiphany system overview, it logically sits between the RDBMS and the Web server.

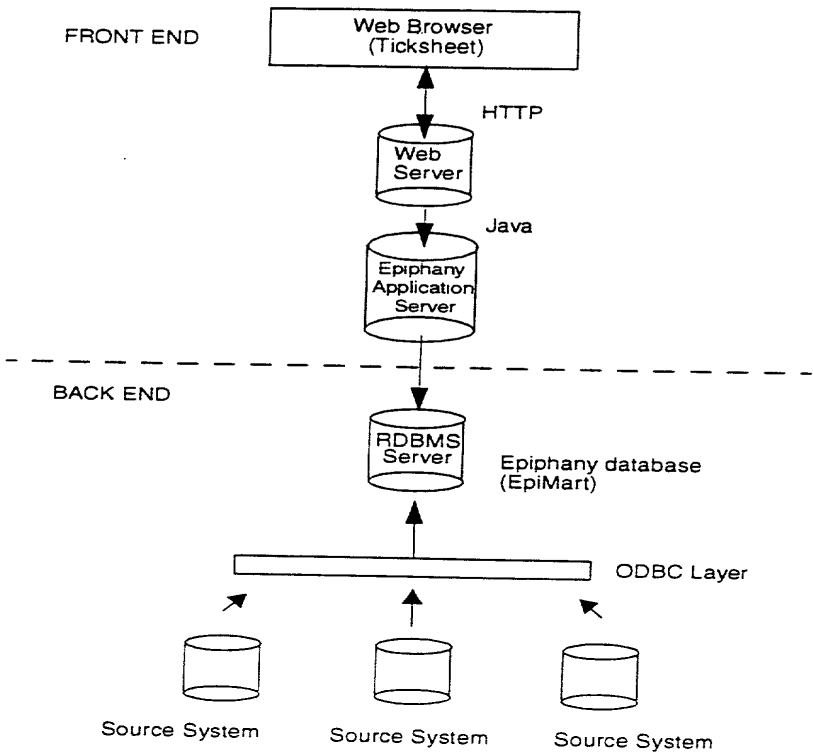


Figure 1-5 Epiphany System Overview

Basic Concepts

Although Epiphany's back-end tools can be used to generate and populate an industry-standard star schema, the power of the system derives from its integration with Epiphany's end-user Enterprise Relationship Management (ERM) applications. In order to achieve a robust, consistent application suite, each component interacts with a common metadata repository called EpiMeta.

EpiMeta is built on top of a relational database engine. EpiMeta's tables store all persistent aspects of an Epiphany implementation. When you configure an Epiphany system (using EpiCenter Manager), these are the tables you write to. Each of Epiphany's ERM applications is a consumer of this metadata, which determines the appearance of the applications, as well as other aspects of their behavior.

09625518.072500

Epiphany Database Extraction

The goal of the extraction process is to extract raw data from data source systems and to place it into EpiMart tables where the data is accessible for query by front-end users. The extraction process, which is usually performed on a nightly basis, involves the use of these components:

- **EpiCenter Manager.** In addition to using EpiCenter Manager to define your site's star schema and constellations, you will use it to define extraction jobs. See Chapter 3, *EpiCenter Manager, for Instructions*.
- **EpiCenters (EpiMeta and EpiMart).** EpiMeta is the metadata database that defines the star schema and the semantics applicable to an organization. (*Semantics* refers to the means by which data is interpreted for an organization in terms its business processes.) EpiMeta also contains data related to the extraction jobs.
EpiMart consists of fact, dimension, and staging tables. The fact and base dimension tables hold the data of the star schema (actual customer data), and the staging tables help to populate them.
- **Semantics** (types/templates and instances). At a high level, a semantic type reflects the semantics of a company's business processes. During data extraction, semantic instances, which are

post-compilation SQL programs, are used to merge data with the proper semantics into data loaded during previous extractions. See *Data Merging* on page 16 and *Semantic Instance Extraction Step* on page 25 for definitions of semantic-related terms.

- **EpiChannel.** This is the Epiphany extraction program that executes jobs and logs job activity. See *Running Jobs: EpiChannel* on page 37 and *EpiChannel Debugging* on page 44 for more information.
- **Database administration tools.** These tools are supplied by the database vendor to administer the physical characteristics of the databases and the interconnections among them. (Please refer to your database vendor's documentation for information about these tools.)

The major topics covered in this chapter are—

- Extraction Phases: An Overview (*see page 15*)
- Data Stores (*see page 16*)
- The Role of Jobs in Extraction (*see page 18*)
- Extractors (*see page 20*)
- System Calls (*see page 26*)
- Aggregate Building (*see page 26*)
- Custom Fact Indexing (*see page 33*)
- The UNKNOWN Dimension Row (*see page 34*)
- Running Jobs: EpiChannel (*see page 37*)
- EpiChannel Debugging (*see page 44*)
- EpiChannel Output (*see page 52*)

- Monitoring Jobs (*see page 53*)
- Mirroring: A and B Tables (*see page 55*)

The first topic addressed is one of the Epiphany system's basic concepts.

Basic Concept: The Epiphany System Is Metadata

All of the control information for an EpiCenter is stored in a single metadata repository called EpiMeta. EpiMeta represents a transactional, fully relational model of over one hundred and fifty tables that have complicated declarative referential integrity constraints. Epiphany provides tools such as Web Builder and EpiCenter Manager for configuring this metadata without the need to write to, or even know about, the underlying data structures.

The EpiMeta tables of metadata control all aspects of the system including:

- Star schema structure
- Extraction workflow and semantics
- Aggregation and other performance enhancements
- Business calculations (or measures)
- User interface layout
- Security information
- Saved Report objects

EpiMart contains the customer data at an implementation. The schema of tables in EpiMart is completely determined by the schema metadata in EpiMeta. The contents of EpiMart tables are determined by the instructions contained in the extraction metadata tables. Theoretically, the contents of EpiMeta alone can be used to reconstruct an equivalent EpiCenter; thus re-extraction from the source system should result in an identical EpiMart. Because EpiMeta effectively defines an EpiCenter, Epiphany provides an export/import utility to back up metadata, or to transfer subsets of metadata between EpiCenters. (see Appendix H, *Export/Import of Metadata*, for more information.)

Epiphany Database Extraction

Each Epiphany tool reads from and/or writes to EpiMeta. The interaction between tools and EpiMeta is described below:

EpiCenter Manager	Writes schema metadata, extraction metadata, security metadata, and aggregate definition metadata.
EpiChannel	Reads schema, extraction, semantic metadata. Writes logging data about the extraction.
Aggbuilder	Aggregation, the pre-calculation of selected facts to speed up the front-end query process, is performed by the Aggbuilder program. Aggbuilder reads schema metadata and aggregate definition metadata, and writes aggregate navigation metadata. <i>Aggregate navigation</i> is the process by which the Epiphany query machinery determines the optimal aggregate table to use to satisfy an application's request for information. (The <i>query machinery</i> is the component of the Epiphany Application Server that communicates with EpiMart and actually issues SQL statements against the DBMS.)
Web Builder	Web Builder is used for configuring user-interface metadata, including measures, ticksheets, and dictionary entries. It reads schema metadata and writes measure and ticksheet metadata.
Epiphany Application Suite	Clarity and Relevance applications read schema, measure, ticksheet, and aggregate navigation metadata. They write saved object metadata.

Each table in EpiMeta defines an integer primary key. The database engine provides the actual unique primary key values for each row in the metadata. All foreign keys between metadata tables are accomplished with these integer

columns. The benefit of using non-natural primary keys is that all other columns in the metadata can be changed without affecting relationships in the model. Note that each foreign key uses declarative referential integrity to ensure consistent metadata. Additionally, EpiMeta enforces other declarative integrity constraints based on the correct interpretation of these tables.

Extraction Phases: An Overview

There are three main phases of the extraction process—load (pull/push), data merging, and aggregate building. Each phase is related to the metadata that defines the tables and their columns.

Load Phase

During the load, or pull/push extraction phase, data is extracted from a source database or file in raw format and loaded into interim tables (staging tables or external tables). A *staging table* is a table that contains all of the fields required by a single base dimension or fact table. Staging tables hold the data in preparation for the second phase of extraction: data merging. An *external table* is a table built in EpiMart (and thus internal to the Epiphany system), which usually serves as a temporary table during extraction. The external table, which is similar to a staging table, receives the data directly.

Staging and external tables have a schema similar to the target EpiMart tables (Epiphany's Adaptive Schema Generator generates both simultaneously). The *Adaptive Schema Generator* is a component of EpiCenter Manager that builds the tables in EpiMart using the schema metadata definitions in EpiMeta.

When you run the EpiChannel program to start the extraction process, it reads the metadata for the job you selected and begins executing extractor steps. An *extractor* logically specifies a series of steps that move data from the input to the output data source. Occasionally, an extractor step is directed to populate external tables rather than staging tables. External tables serve as intermediary tables when more complicated multi-staged extraction is required.

Data Merging

The second main phase of extraction is data merging. After the staging tables are fully loaded, semantic instances merge the new data in the staging tables into the EpiMart database tables. A *semantic instance* is the usage of a generic semantic type on one fact or dimension table during part of an extraction job. As part of defining tables, you assign each an Epiphany-defined semantic type. A *semantic type* refers to the logical business process for which a generic SQL program called a *semantic template* will be applied.

The Epiphany extraction program, EpiChannel, executes the semantic instance at the appropriate time during an extraction job.

Aggregate Building

The third and final extraction phase is aggregate building. Aggregates are pre-calculated and pre-stored summaries of data that significantly accelerate the front-end *query machinery*. The query machinery is the component of the Epiphany Application Server that communicates with EpiMart and actually issues SQL statements against the RDBMS.

You will use EpiCenter Manager to define which facts are aggregated for groups of dimensions. The Aggregate Builder program, which typically runs immediately following the data extraction, performs roll-ups, or aggregations, on the groups and includes them as aggregates in the EpiMart tables.

At the conclusion of the extraction phase, the extracted data resides in the proper tables and columns in the EpiMart. The data has been extracted from the source system or file data source (called a data store) and placed in the data store on the RDBMS server.

Data Stores

A *data store* is a logical location of data that functions as either as a source or sink within an EpiCenter. Data stores include relational database connections, as well as files and directories. The concept of a data store is integral to the Epiphany extraction process. A data store represents the physical location of a source of data that the Epiphany system can read from and/or write to.

09625518 1072500

Typically, the input data store is a source system database or file, and the output data store is EpiMart, which holds customer data, on the RDBMS server. As part of the installation procedure, you will create a new, empty database for EpiMart (and EpiMeta).

As shown in Figure 2-1, the Epiphany system extracts the raw data from a site's source system—for example, a RDBMS server, a generic ODBC source system, or a flat file—via Open Database Connectivity (ODBC). ODBC is an abstraction layer that provides a common interface to many databases. This data is read by various ODBC drivers and written using the target database system's API. The Epiphany system can access data if an ODBC driver exists.

EpiChannel, the Epiphany extraction program that executes jobs and logs job activity, opens an ODBC connection to the source system and a database library connection to the target server and initializes tables.

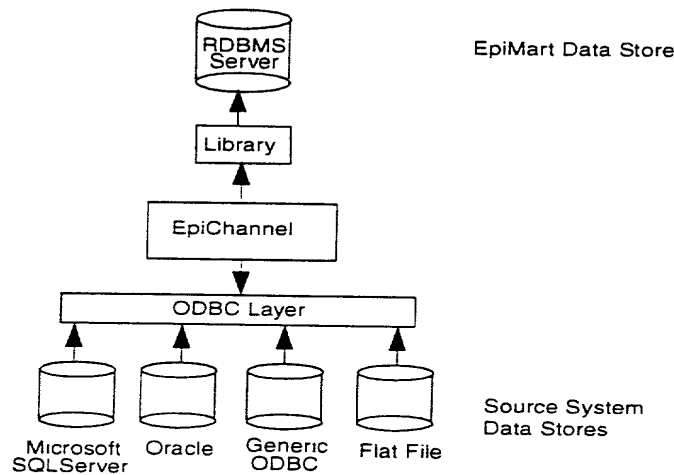


Figure 2-1 The Back End of the Epiphany System

You will use EpiCenter Manager to configure the data stores as part of setting up your site's extraction process. When an EpiCenter is created, it has three default data stores:

- *EpiMart*
A special data store for the actual database on which Epiphany's front-end applications will run.
- *JobFileLog*
Specifies the data store for EpiChannel job logs.
- *LoggingDB*
Specifies the data store for the EpiChannel logs.

You may, however, set up as many data stores in EpiCenter Manager as necessary. For example, operational systems for different divisions within an EpiCenter may have the same fact and dimension tables (EpiMart), but draw data from different source systems. In this case, you would create a data store for each input source system. For more information, see *Data Stores* on page 90.

Note: The EpiMart is the data store, or database, that you set up for your data warehouse. The EpiMeta holds the description of your data (the extraction jobs, the ticksheets, and the star schema definitions). The EpiCenter is a combination of EpiMart and EpiMeta.

The Role of Jobs in Extraction

The Epiphany extraction process works through the running of jobs. A site defines the jobs needed to extract its data and runs these jobs on a regular basis to update the EpiMart. A job is basically a batch of work to be performed as a unit. The work consists of an ordered list of job steps, each of which is either an extractor or a system call. System call job steps may be interspersed with extractor job steps in any order. (See Figure 2-2, page 19 and Figure 2-3, page 21.)

A simple extraction may consist of one job, which has multiple steps. Each step utilizes the data produced by the previous step. A multi-stage extraction, however, may consist of multiple jobs. The EpiCenter Manager provides a graphical user interface (GUI) for defining job steps and the job input and output data stores. The actual job steps (extractors and system calls) need to be customized for each site.

After the jobs have been defined and the EpiCenter schema generated, the database administrator can invoke the EpiChannel (**extract.exe**) command to begin the extraction process that will populate a new EpiMart, or update an existing one.

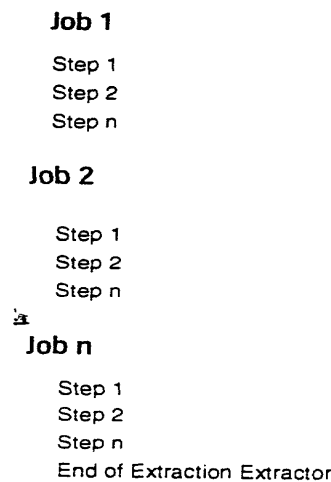


Figure 2-2 Job Work Flow

Extractors

An extractor is one or more sets of SQL operations that will be executed against a particular source and destination data store. These same sets or groups of SQL operations may be shared by another extractor that applies them to different data stores. The SQL operations may be either SQL statements or semantic instances. In general, SQL statements are used to extract data, and semantic instances are used to merge data with the proper semantics into data that has been loaded during previous extractions.

Semantic instances are designed to work on an idealized star schema. Macros in these semantic instances are translated at extraction time to the tables and columns in the target star schema. (The fact or dimension table listed in the Semantic Instance dialog box (Figure 3-20, page 101) in EpiCenter Manager determines which column names are actually used in the final SQL.) This enables the complex transformation logic in a semantic instance to be applied to star schemas other than the one for which the semantic instance was first used.

As shown in the job work flow (see Figure 2-3), a job step may be either a system call or an extractor. An *extractor* is a list of extractor steps (and input and output data stores) that move data from the input to the output data store. An *extractor step* is a single step of an extractor, which always points to an extraction group. An *extraction group* is a set of extraction steps. An *extraction step* is a single atomic extraction operation that can be either a SQL statement or a semantic instance.

Extraction steps and groups are both modular. After you define them, you may reuse them in other jobs. (The Extractor Steps dialog box (Figure 3-18, page 97) in EpiCenter Manager is where you define a new, or select an existing extractor for a job.)

Note: An extractor has associated input and output data stores and exists independently of any extraction step.

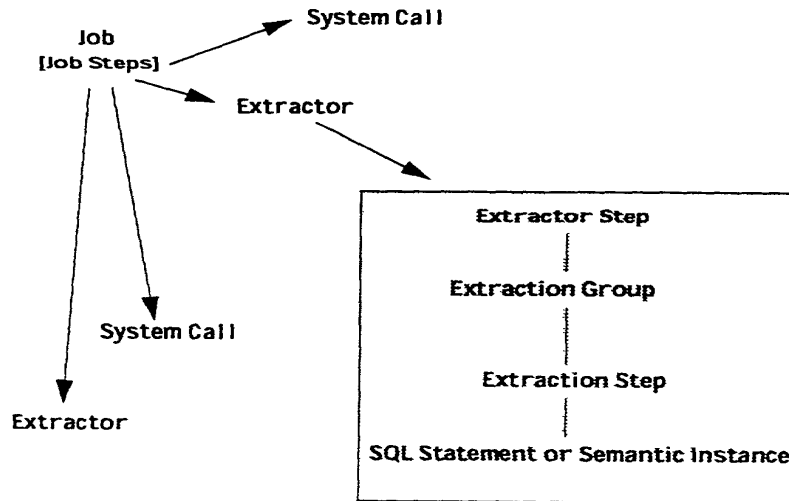


Figure 2-3 Job Structure

An extractor can move data from a single source data store to a single destination data store. A job may consist of multiple extractors, however, each with its own independent input and output data stores.

Jobs may also share extractors, applying the same logic in different orders—if the jobs' data stores have common structures. For example, a job that runs

Monday through Friday could share the same extractor with a Weekend version of the same job in which the only difference is that the Weekend job has an extra step, an extractor that updates rarely changing tables.

The two kinds of extraction steps that may comprise an extractor step are discussed below: SQL statement and semantic instance.

SQL Statement Extraction Step

An SQL statement extraction step may be either a stand-alone SQL statement or a pull/push statement.

A stand-alone SQL statement is executed against either the source or destination data store for that extractor and is usually specific to the data store's type. These SQL statements are usually issued to achieve a side effect, such as dumping the transaction log or setting up environment variables. Any returned results are discarded.

A pull/push SQL statement is a type of extraction step that issues SQL against an input data store (and is interpreted according to the rules of that database's engine). The returned results are inserted (pushed) into the destination table for that step, either a staging table or an external table (the output data store of the extractor).

The result set columns are mapped either to columns in the destination tables or to other functions. The mapping is a case-insensitive exact match by name of the available columns, or function results, to the destination table columns.²

At this stage of the extraction, each destination table column must have a matched value. If not, the mapping fails, the SQL statement results are discarded, and the statement fails. Extra columns that are available from the SQL statement or functions, but which are not used in the destination system, are not considered to be errors. (SQL statements with extra columns will have their mapping displayed in the EpiChannel log if you set the verbosity level for EpiChannel high enough; see *EpiChannel Debugging Levels* on page 50.)

². Special characters such as the *at* symbol (@) are removed when this match is performed.

Once mapped, the rows from extraction SQL statements are fetched into EpiChannel memory where they are verified for field width and forwarded to any function that consumes the field. The mapped results are forwarded to the destination database.

All operations on the source data store or the EpiMeta data store occur through the appropriate native library for the EpiCenter. The inserts are batched so that the extraction statements maintain a count of rows fetched, rows forwarded, and rows committed. The batch is fully sent and committed before the extraction SQL statement ends and the next job step may begin.

Any error, as determined by the database engine evaluating the SQL, results in the SQL statement being considered in error. If the action plan for the SQL statement (as selected in the General tab of the SQL statement dialog box) is to abort, then the job halts without executing any other SQL operations or job steps.

SQL Macros

A SQL statement may contain extraction macros that are expanded before the SQL is executed. Some of these macros provide database-independent functionality by expanding in different ways for different databases, and some use the metadata to cause the SQL to reflect the edits made in EpiCenter Manager. These macros are described in Appendix B.

Staging Tables

The most common destination for an extraction statement is a staging table. As mentioned, a staging table is a metadata table that contains all of the fields required by a single base dimension or fact table. Unlike the real fact and base dimension tables in the EpiMart, the staging tables have dimension columns that contain key fields from the customer's system, rather than Epiphany-generated foreign keys. Staging tables typically contain data extracted by only one job, rather than accumulating data over time.

Once a staging table is populated, semantic instances are used to merge the new data in the staging table into the existing data. A semantic instance is a series of statements, similar to a program. These statements maintain the integrity and additive nature of the columns in the facts tables and maintain correct references to dimensions even though the dimension may change from one reporting of a fact to another.

See Appendix F, *Writing Staging SQL Statements*, for more information.

External Tables

Extraction statements are sometimes directed to load external tables rather than staging tables. External tables serve as intermediary tables when more complicated multi-staged extraction is required, such as for the grouping of data, dividing values in a field into bins, or joining with data from another source system. They may also serve as holding fields used by non-Epiphany data transformation tools, such as third-party name and address cleansing tools. In the case of cleansing tools, extractors merge the cleansed names and addresses with other tables extracted from the source databases into the EpiMart tables.

EpiCenter Manager provides the extractor named *End of Extraction* for each EpiCenter by default. This extractor consists of two steps:

1. Issue SQL against the EpiMart to determine the date of the last extraction to be displayed to users.
2. Toggle the current EpiMart from A to B, or vice versa.

Step 1 usually consists of SQL statements that load the external table named *last_extract_date*. The *last_extract_date* table is provided by Epiphany, and is recommended for use at your site.

Step 2 updates the *config_master* value *last_extract_date* with the latest value received from Step 1. It then updates the *config_master* value called *current_datamart*, switching it between A and B.

Note: The last extraction date setting is the date that appears as the *Data is valid as of...* in Clarity reports. It is determined by an option in the EpiCenter Manager's Configuration dialog box (described in Appendix C).

Switching the focus between the A and B tables involves an update to a single row in the database (no tables or databases are actually moved). See *Mirroring: A and B Tables* on page 55 for a discussion of A and B tables.

See *Using External Tables as Inputs to Staging Queries* on page 266 for more information about external tables.

Semantic Instance Extraction Step

As mentioned, the Epiphany extraction process (exclusive of aggregation) is a two-phase operation:

- load (pull/push) phase
The load phase refers to the loading of staging tables, which are the first entry points of data into EpiMart.
- data merging phase
The data merging phase refers to the loading of staging tables into the EpiMart's permanent base tables.

This process of moving data from staging tables into the EpiMart's base tables is called *semantic transformation*. Proper configuration of the data merging phase of extraction, requires an understanding of the underlying meaning, or semantics, of the data.

Epiphany provides a set of semantic types/semantic templates that you may use without having to be concerned with their internal code. A *semantic template* is a generic SQL program intended to accomplish specific business rules during extraction (these rules are referred to as *semantic types*). A semantic template program contains SQL with generic table names; it does not refer to actual column or table names. Only when the semantic template is applied to a base dimension table or fact table does the SQL contained within it refer to actual column and table names.

When a semantic template is combined with a base dimension table or a fact table, the result is a semantic instance. A *semantic instance* is a valid SQL program that can be run against EpiMart.

Applying a Semantic Type

Epiphany provides a set of semantic types for each EpiCenter. When you use the Semantic Instance dialog box in EpiCenter Manager to assign a semantic instance, you select one of the semantic type options. The list of available options is determined by whether the object that the semantic instance will operate upon is a base dimension table or a fact table. Your selection assigns the semantic type to the table (a row is added). The implementation, as opposed to the assignment, of a semantic type is stored in metadata tables and is invoked by EpiChannel at run time.

Dimension semantic types have two purposes: creating base dimension tables and creating dimension mapping tables. Dimension mapping tables are used to convert dimension source system keys (*sskey*) in fact tables to Epiphany dimension keys. Dimension semantic types are also responsible for proper indexing of these two tables.

Fact semantic types are normally used to populate and Index new fact base tables. For descriptions of the fact and dimension semantic types, see Appendix G. See Appendix F, *Writing Staging SQL Statements*, for a description of source system keys.

System Calls

While SQL-based transformations may be sufficient to extract from simple databases, sites with more complex databases may require multi-stages and additional commands, such as lookup tables, aggregation splits, gathering data into ranges (binning), and duplicate detection. For this purpose, you may use system calls, which are executed during a job as if invoked from the console's DOS command line.

In a multi-stage extraction, files might need to be moved, decompressed, de-encrypted, or purged. Although these steps could be placed before or after extraction statements, they may need to be placed in-between SQL-based steps. EpiChannel coordinates the execution of system calls between extractors.

A complex job might require the following steps:

1. The invocation of a system call to uncompress a file.
2. The invocation of an extractor to populate external tables.
3. A system call to invoke third-party software to cleanse the names in the external tables.
4. Extractors to merge the cleansed names with other tables extracted from the original and other databases into the EpiMart.

Many system calls need to reference the location of databases or files. In the above examples, Steps 1, 2, and 3 need to share the file and database names for the data they pass to each other. Although these names could be hard-coded, this is not the best approach; for example:

- The system call could break if the database login information changed, and the system call text was not modified accordingly.
- The system call cannot be easily reused.
- Some file names need to be adjusted from run to run so that useful intermediate data is not overwritten.

Job data store roles provide an alternative to hard coding database and file information in system calls. Roles are referenced by macros in systems calls, and these macros look to the job to see what data stores are associated with the roles. The information from the matching data store is replaced by the macro. For example, the system call

```
mkdir $$DIRNAME[Working Dir]/aggbuilder_temp
```

would have the `$$DIRNAME` macro expanded to the MS-DOS file path that EpiChannel is using as a *working directory*. The **mkdir** command then creates a subdirectory in this location. Because EpiChannel generates a unique directory name for itself for each run, the above command would create a corresponding unique subdirectory for use by the Aggregate Builder program (Aggbuilder), and this directory would be moved/purged along with EpiChannel logs.

See Appendix B for more information about Epiphany macros.

Note: All system calls are expected to have an exit code of zero, but you can use the Add Job Step dialog box (see Figure 3-23, page 106) in EpiCenter Manager to set the *On Error* type to Abort or Ignore, as appropriate.

Aggregate Building

Creating aggregates speeds up system performance at the front-end of the Epiphany system. This is because some queries simply require summary data, which can be pre-computed by the system rather than being recomputed for each query. For example, assume that a site has 5,000 products categorized by ten product types. Clarity users may request reports that show data aggregated by product type. If there is a grouping by product type, the Epiphany aggregation program creates a table with ten rows (one for each product type) and pre-calculates “rolled-up” (combined) fact aggregates based on this smaller dimension table. By using significantly smaller fact aggregates, the system can generate reports considerably faster.

When EpiMart is successfully populated, the rows in the fact tables contain foreign keys to dimension table rows. In Epiphany, all dimension primary keys are stored as integers. For instance, fact rows may resemble the following in the fact table Order_O:

Order_O

Customer_key	Product_key	Salesperson_key	Total Sale
6	8	5	\$20
5	6	6	\$30

whereas the dimension tables may look like these:

Customer_O

Customer_key	Customer Name	Region
5	ABC Company	West
6	Bill's Lighting	West
7	Fred's General Store	East

Product_O

Product_key	Product Name	Platform
6	Product A1	Windows
7	SuperX	Mac
8	Smash-Em	Windows

Salesperson_O

Salesperson_key	Salesperson Name
5	Gene
6	Sue

All fact and dimension tables that end with _O (underscore zero) are called base tables in EpiMart. They contain raw data at the level that is extracted from the source system.

A base dimension is a physical dimension table in EpiMart, such as Customer or Product. Base dimension tables contain the actual dimensional values that are available for reporting or filtering. Attributes, such as Customer Region or Product Platform, can be rolled up. For example, end users often want to create reports of total sales by region. Suppose that Order_O has hundreds of

thousands of records, and Customer_O has many thousands of records, and that there are only three different regions in the Region field. A report of sales by region against Order_O and Customer_O will need to aggregate over thousands of records, causing the report to return answers very slowly. However, a pre-built aggregate with three (rolled-up) rows that already has the total sales by regions could answer the same question very quickly.

After configuring an EpiCenter, the following aggregates might be built:

Order_17

Customer_key	Total Sale
1	\$50

Customer_1

Customer_key	Region
1	West
2	East

Note the following:

- The *Customer_1* table contains only the rolled-up field(s). The number of rows has been reduced so that each row contains a distinct Region. The table name is the same as the base table, with a different number at the end.
- The *Order_17* table contains the same total sales dollar amount, but the number of rows has been reduced (in a larger example, this reduction could be extremely significant). The name of the table is the same as the base table, with a different number at the end.

- The foreign key *Customer_key* has the same name as in the base tables above, but the numbers differ. It is important to recognize that the *Customer_key* field in *Order_17* can only be joined with the *Customer_1* table, since *Order_17* is at the granularity of Regions, not Customers.

See *Defining Dimension Aggregates* on page 71 for a description of how tables such as *Customer_1* are built using EpiCenter Manager.

The Aggbuilder Program

Aggregate building is performed by Aggbuilder (**agg.exe**), a program similar to EpiChannel. Aggbuilder uses the same command line, EpiMeta database, and job definition table as EpiChannel. Aggbuilder, however, completely ignores the job steps (that is, system calls and extractors). It uses the job table only for an indication of the *log database* and *working directory*. The structure of Aggbuilder log files is similar to that of EpiChannel except that the *log* directory name is *agg_timestamp* instead of *chn_timestamp*.

The easiest way to invoke aggregate building is via a system call that uses some of the system-call macros to isolate details. To invoke Aggbuilder, add a system call similar to this after the extraction and semantic instances steps, but before any use of the extractor called *End of Extraction*:

```
$$AGG $$EXC_ARGS -j agg_job_name
```

where *agg_job_name* is the name of the job with the proper logging information.

To log to the same places as the current job, use a system call such as:

```
$$AGG $$EXC_ARGS -j $$JOBNAME
```

The Aggregate Building Process

When the Aggbuilder program is executed during an extraction job, the following actions are taken:

Note: All actions occur in the set of mirrored tables (A or B) that is *not* currently active. See *Mirroring: A and B Tables* on page 55 for more information.

1. Dimension aggregates are built. Each column set for a base dimension will become a dimension aggregate as long as that column set is included in an aggregate group via EpiCenter Manager's Aggregate Group dialog box (see *Aggregation and Aggregate Grouping* on page 81). Indexes are built for these tables as appropriate.
2. For each fact table, all enabled aggregate groups to which that fact belongs are examined. Aggbuilder determines all possible combinations to be built based on the dimension role definitions and creates unique table names. Fact aggregates are physically built and indexed.
3. Aggbuilder records what it has built in a set of read-only metadata tables in EpiMeta. You may use EpiCenter Manager to browse these lists. The Epiphany Application Server uses these same lists at runtime to decide which aggregates to use to satisfy an end-user query.
4. All activity is logged to Epiphany's logging tables.

Custom Fact Indexing

EpiCenter supports two methods that dramatically improve query performance: indexing and aggregation. Each accelerates different classes of queries.

In general, the presence of aggregate tables accelerates queries that answer high-level business questions, such as Sales by Business Unit and Fiscal Year. Deep drill-down queries, or highly filtered queries that ask for a small subset of data, are typically enhanced by an index. A fact base table is usually the largest table in EpiMart. Even tables with millions of rows can be accessed quickly via an index when only a small number of data pages is needed to service a query.

A given Epiphany query uses only a *single* fact table (either base or aggregate). Typically, as a user drills down into a result set, the physical table that services each query moves from a high-level aggregate (with few rows) to a larger aggregate (possibly using an index on the aggregate). For the user's response time to be acceptable in each of these cases, you must properly configure both aggregates (as described in the previous section) and custom indexes.

All aggregates are indexed in the same manner as the base table (insofar as the aggregate has the same dimensionality as the index). For example, if the base table has an index on *customer_key*, *product_key*, and *territory_key*, then an aggregate that contains only the Customer and Territory dimensions will have an index on *customer_key* and *territory_key*.

Note: If two different indexes on the base table result in the same index on the aggregate, then the aggregate index is built only once.

By default, each fact table has one index built on its dimensional keys during extraction (on Microsoft SQLServer this index is also the CLUSTERED index). For all fact tables, the leading term of the index is *date_key* because a wide class of queries filters on the time dimension. For example, if a system contains data for multiple years and a user asks for a single calendar month, the database almost certainly will use an index when it services this query (whether via the base table or an

aggregate). All other dimension roles in the fact table are also included in this index. The order in which they appear is determined by their order in the constellation within EpiCenter Manager's directory tree.

To re-arrange this order, right-click a dimension role folder, and select Up or Down from the pop-up menu.

A custom index is the metadata definition of indexes to build on fact tables in EpiMart. Normally, each fact table is given a single index. You can use EpiCenter Manager (the Custom Index tab in the Fact Table dialog box) to:

- configure all other indexes on the fact table.
- assign each index a logical name, although the name is not used when building the physical index.
- choose the dimension roles that make up the index, and their order.

Note: You will use the Custom Fact Index semantic type to actually build the indexes. (See Appendix G, *Semantic Types*.)

It often makes sense to build several different custom indexes, each with a different dimension role as the leading term. If, for instance, end users often filter on attributes of the Customer and Product dimensions, then you could build a custom index on the combination (*customer_key*, *product_key*) and another on *product_key* by itself. There is no need for the second index to also include *customer_key* because the first index could easily service joint customer/product queries.

The UNKNOWN Dimension Row

When a fact staging table is loaded during an extraction, its dimension source system key columns (columns whose names end with *sskey*) normally refer to rows in the base dimension staging tables. (The column name being referred to in the base dimension staging table also ends with *sskey*.) These source system keys are the actual values used to form the relationships on the source system between the fact and dimension source tables.

Some source systems do not enforce referential integrity, however, which may result in source system fact tables that have “dangling references,” or references to non-existent dimension values. Another possible source of a dangling reference is an optional or null foreign key in the source system. In EpiMart, all fact dimension keys must point to valid base dimension entries; and there should be no dangling references.

To prevent dangling references, each base dimension in EpiMart is populated automatically with a single special row called the UNKNOWN row. This row is always available for mapping fact dimension keys that would otherwise not point to a valid base dimension row.

When writing fact staging SQL statements, you can refer explicitly to this special row by populating the *sskey* dimension role column with the special string UNKNOWN. For instance, on a Microsoft SQLServer source system, you might use the ISNULL() function as follows to translate all null values to UNKNOWN:

```
SELECT
    ...
    ISNULL(cust_code, 'UNKNOWN') customer_sskey
    ...
FROM
    ...
```

Referring to the UNKNOWN Row during Extraction

For a base dimension such as Customer that has dimension columns *customer_name* and *region*, the UNKNOWN row has actual values similar to any other row. By default, each dimension column assumes the literal value UNKNOWN for each of its textual attributes. However, you may use EpiCenter Manager to override this default value so that end-user queries return a different value. To do so, enter a literal string (without quotes) in the Dimension Column dialog box (accessible from the General tab of the Base Dimension dialog box shown in Figure 3-6, page 70). Your string will be used instead of UNKNOWN.

The Update Unjoined Semantic Type

While extracting a fact staging table, one often does not know whether the values that appear in the dimension foreign key columns actually refer to dimension rows. While the technique described above for converting null values to the UNKNOWN string works for missing dimension foreign key values, it will not work when a dangling value appears. If no action is taken, then these fact rows will be lost during extraction since the first step of most fact semantic types involves an inner join between the fact staging table and the dimension mapping tables (see Appendix G, *Semantic Types*).

The Update Unjoined fact semantic type is specifically geared towards converting these dangling fact references to the special UNKNOWN value. Scheduling this semantic type (after the fact staging table has been loaded) during an extraction will cause a scan of the fact staging table with joins to each of the dimension mapping tables. All *sskey*'s in the fact staging table that do not map to extracted dimension rows will be converted to UNKNOWN.

Note: Execute the Update Unjoined fact semantic type before executing any other semantic types on this fact staging table.

In general, it makes sense to execute semantic types for base dimensions before any of the semantic types for facts (including Update Unjoined) because the base dimension semantic types produce the mapping tables needed by fact semantic types (see *Normal Extraction Order* below).

Normal Extraction Order

Epiphany's recommended sequence of events during an extraction is as follows:

1. Load fact staging tables.
2. Load base dimension staging tables.
3. Execute base dimension semantic Instances.
4. Execute fact semantic instances.

The reason for loading the fact staging tables before base dimension tables is subtle. If Epiphany is extracting from a live source system, then new dimension and fact rows could be created while the extraction occurs. If dimensions were extracted first, then when facts are extracted, a new dimension row (such as a new Customer) might have been created, but was missed. If a fact that refers to that new Customer is then extracted, the fact row shows up in the Epiphany system with an UNKNOWN Customer value (because that Customer row was missed by the extraction). By extracting the facts first, the system might not receive the fact row in this extraction, but will retrieve it in the next extraction.

Running Jobs: EpiChannel

EpiChannel is the Epiphany extraction program (**extract**) that you will run to populate your initial EpiMart and to update it on a regular basis.

EpiChannel first opens an ODBC connection in the source system and then opens a database library connection to the target system and initializes tables. It establishes start and stop date time boundaries and replaces certain special strings. After replacing these strings, EpiChannel executes a user-defined sequence of SQL statements or system calls.

The SQL is pre-processed to match warehouse and database vendor syntax. System calls are pre-processed to supply directory locations and database login information.

Before attempting the first job step, EpiChannel verifies the availability of databases and tables (if these options are selected in EpiCenter Manager's Job dialog box) and expands most macros. If these fail, the job does not run.

The job will halt when it encounters the first error in any step, unless that step has an error code set to print or ignore the error instead of to abort it. Error codes are set through EpiCenter Manager. You will use the SQL Statement dialog box (Figure 3-19, page 98) for extractors, and the Job dialog box (Figure 3-21, page 102) for system calls.

Whenever a job succeeds or fails, e-mail notification is automatically sent to the database administrator (or any designated list of e-mail recipients).

The EpiChannel Command Line

This section describes the command-line syntax you will use to invoke EpiChannel at the console's DOS prompt. These command-line parameters indicate the database with the job information (that you configured using EpiCenter Manager) and parameters related to interactive debugging. Command-line parameters may also specify the initial debugging level, the maximum selection limit, or that the jobs are to be trial runs.

To invoke EpiChannel, enter the **extract** command on the console's DOS command line followed by the parameters described below.

Note: EpiChannel inherits the case sensitivity of the database engine.

extract

```
-h
-?
-v integer_verbosity_level
-I instance_name
-S server_name
-D db_name
-B db_vendor_name
-U username
-P password
-t
# row_count
-j job_name
```

where:

- h and -?** Display detailed on-line help.
- v *integer_verbosity_level***
Runs the EpiChannel job in verbose mode, which displays the SQL on the screen. See *EpiChannel Debugging Levels* on page 50 for more information.
- I *instance_name*** Specifies which Registry instance key to use. This parameter is mutually exclusive with many of the other parameters because it causes them to be read from the Registry.
- S *server_name*** Specifies the name of the server where the EpiMart resides.
- D *db_name*** Specifies the name of the EpiMeta database.
- B *db_vendor name*** Specifies the name of the EpiMart database vendor.
- U *username*** Specifies the username for the EpiMeta database.
- P *password*** Specifies the password for the user of the EpiMeta database.
- t** For testing purposes, this is the trial-run option that simulates execution of the SQL on the source and destination databases.
- # *row_count*** Specifies the maximum number of rows to fetch on any SQL statement.
- j *job_name*** Specifies the name of the Job as defined in the EpiMeta.

EpiChannel Registry Keys

The EpiChannel program uses the Registry keys shown in Table 2-1, which are located in the following Registry path on the local machine:

Software\Epiphany\Instances instance_name.

In the table below *Default Instance* is the string value for Default in the Instances Registry key under the Epiphany Registry key.

Table 2-1 EpiChannel Registry Keys

Key	Value
1	Default Instance
2	[Default Instance]/DBVendor
3	[Default Instance]/Database
5	[Default Instance]/Server
6	[Default Instance]/Username
7	[Default Instance]/Password

By default, EpiChannel reads the Default Instance key and uses its value to open the Registry tree that holds all initialization parameters. You can override this by specifying a different name using the *-I (instance_name)* option.

For example:

extract -I name

forces EpiChannel to read all Registry values from
HKEY_LOCAL_MACHINE\...\Epiphany\Instances instance_name.

Other EpiChannel command examples:

- A command (in a system call) in which all Registry variables are set:
`extract -j job1`
- A command example in which no Registry variables are set:
`$$EXC_CMD`

Output Files

Jobs may log to files or databases, and these logs can be directed to multiple destinations simultaneously. For example, one log could be placed on the local machine and another on a network server.

The execution of jobs is logged as a unit. Statistics are collected for job steps and for entire jobs. The location of the log is established by associating data stores with the *log* or *working directory* roles for a job, for which you will use the Job dialog box in EpiCenter Manager (see Figure 3-21, page 102).

Extracting New Rows Only

An efficient extraction process pulls only data from the source system that has changed or been added since the previous extraction. The way that the extraction process recognizes this subset of data depends on how the source system identifies changes in data. Epiphany supports the following methods of identifying changed data:

- Date or Date/Time fields in source rows
A source row, or a table it joins to, contains a date or date/time indication of when it was inserted or last updated.
- Timestamp fields in source rows (Microsoft SQLServer only)
Values can be “stamped” in some way to indicate their status. Epiphany provides Microsoft SQLServer timestamp fields for source rows. A source row, or a table it joins to, has a column that contains Microsoft SQLServer data type *timestamp*. The database engine automatically updates this column whenever the row is inserted or updated. This field does not

actually contain any record of the time that the update occurred, but rather values that always increment, such that any row edited after another row will have a higher timestamp.

- Highest numeric or string value in named columns in source rows

A particular column in a particular table is presumed to contain a value that is always increasing from row to row; for example, Order Number or Policy ID. For this method to be useful as a subset selection criterion, however, either the rows must never be updated, or else the column must be changed by the source system whenever an update occurs.

How EpiChannel Identifies Data To Be Extracted

The following topics describe how EpiChannel identifies which subset of data to extract:

- How EpiChannel reads values

EpiChannel attempts to get a consistent set of dates/timestamps/column values so that the same single moment in time is captured by the WHERE clauses generated by the extraction set identification macros. The extraction program minimizes the chance for changes to occur to one set of values, and not to another one, by reading all sets one after another with no intermediate actions. While this gives highly consistent values, it is possible that a change may occur between the time EpiChannel reads the first table/column and when it reads the last one, especially if multiple databases are involved.

The values read are stored as the “last” values only if the job commits its work. If any system call or SQL statement has an error that aborts the job, it is as if the value stamps had never been read or modified. In the unlikely event that a job aborts during the commit phase, the timestamps might not be synchronized with each other.

The date and timestamp values are read from a table-independent current value such as SYSDATE or GETDATE().

- Extraction subset extraction

Epiphany supports *extraction subset extraction* by reading and recording the current values of these fields at the start of the run and remembering these values for the next run. The values are made available in SQL through the SQL macros described in Appendix B.

- Ignore Timestamps option

The Ignore Timestamps option, which you can set using EpiCenter Manager's Job dialog box, causes all of the value range expressions to become (1=1), or true, which means that all data is selected without regard to its value range.

- "Priming" value range memory

The value range memory needs to be "primed" by a trial run whenever you add a new range expression. Basically, the first run causes the memory to start tracking the values for the next run, but all ranges are 1=1 for the current run. The second run causes the "current" values to be read, but all ranges are 1=1 because there are no last values. The third and successive runs have both current and last values, and the range expressions are valid. If you have changed the SQL and are in doubt, make a trial run, which will prime the value memory with any new value expressions.

- Disabling/enabling an expression

If you remove an expression and later return it, the first run after the expression is back picks up all values since the last time the expression was enabled. If you disable a SQL statement that contains a column value expression and no other SQL uses that same column value, the column value from the last enabled run of that SQL will be used as the last value whenever the SQL is re-enabled.

Disabled expressions related to dates or timestamps will pick up the last value that any expression found. If all expressions using dates and timestamps are disabled, they will pick up where they left off when they

are again enabled. If any expression using a date or timestamp remains enabled, it “bumps” the date and timestamp memory, and all expressions will pick up the new values when they are again enabled.

- Memory for last values

The “memory” for last values is associated with an extractor and not a SQL statement or a job. If the same extractor is used in different jobs, the jobs will pick up each other’s value updates. For example, the Weekend job will see the values updated by the Weekday jobs if they share extractors.

If the same SQL step is used in two extractors, it will have different “last” values for each extractor, which is desirable because the extractors may access different databases that have different highest values and times.

EpiChannel Debugging

The EpiChannel debugging mechanism is thorough and well annotated. If you set a high enough verbosity level, the complete anatomy of the job is revealed in real time at the console, or historically in the log file. The EpiChannel **-v** option controls the verbosity level. (Running **extract** with the help (**-?** or **-h**) option lists and defines these verbosity levels.)

Job Output

You may display EpiChannel output for debugging purposes on a screen or direct it to a file. The text is displayed on the screen in fixed fonts, such as Courier. The number of characters that can be displayed per line is determined by the number of columns. (You may use EpiCenter Manager’s Job dialog box to set the job log width to 80 or 132.) The output is designed to be adequately displayed on low-end monitors.

At the top of the debugging screen/log, you will find the names and directory locations of the job’s EpiMeta database, as well as its *job log*, *log file*, and *log database*. (See Figure 2-4 for the start of an output file and Figure 2-5 for the log summary.)

Typically, the job begins with the truncation (deletion) of old staging tables and the clearing of indexes. The processing of each step is displayed sequentially; for example:

1.1 Processing step 'PreMfg' as an extractor

followed by the extractor's source and destination databases. Steps that are not enabled are numbered, but not shown. In the job output shown in Figure 2-4, Step 1.1 was disabled.

Each operation follows on a separate line. You can identify the type of operation by the letters or symbol(s) that precedes it:

- **St** A set of SQL statements. The members of this group can be either stand-alone SQL, extraction statements (pull/push), or semantic instances (data merges). Steps may be one of these types:
 - < Stand-alone SQL applied to the source data store.
 - > Stand-alone SQL applied to the destination data store.
 - <> Pull/push SQL that transfers data from the source data store to the output data store.
- **SI** Semantic instances.
- **Sy** System calls.

The EpiChannel output provides summary information that is based on the statement type. The summary information is prioritized and displayed in columns to the right of the operation. If there is not enough space, only the highest priority information is given.

For SQL pull/push statements, the output includes the time taken for the pull/push, the number of rows in the table before the operation, the number of rows transferred, and the number of rows in the table at the end of the procedure. If the log file's width is 132, the number of bytes is also shown.

Stand-alone SQL statements have time as their only metric. Semantic instances define their own metrics.

To conserve space, the summary information is labeled by two-digit codes (tokens) that appear to the right of their numeric values. These tokens are defined as follows:

- **Ti** Time taken, rounded to the nearest second.
- **St** Rows present in the table at the start.
- **Ex** Rows extracted.
- **En** Rows present in the table at the end.
- **UN** Rows unjoined. These fact table rows contained references to dimension values not present in the EpiMart or in the newly extracted data.
- **PR** Rows processed; a count of the gross number of rows examined.
- **IN** Rows inserted; a count of the rows that were successfully moved into the EpiMart. Rows are sometimes omitted because they are earlier than previously recorded data, or because they are redundant to data present elsewhere in the same staging table.
- **MO** Rows modified. Modifications typically adjust references to missing dimension values such that they become references to the predefined UNKNOWN dimension value.

For example, the lines:

Statement Name	Time	Metric1	Metric2	Metric3
St wb bump the test day				
< wb bump the test day				
<> appl real		0 St	3 Ex	3 En
SI Product	1 TI	3 IN	.0 MO	3 PR

indicate the following:

- A set of statements called wb bump the test day was called.
- There is no summary information for this set.
- An execute-only SQL statement called wb bump the test day was run against the source system and took less than a half second to execute, and therefore has no time statistic.
- An extraction statement called appl real inserted 3 rows into a previously empty table and took less than half a second to execute.
- A semantic instance called Product took about a second to execute and processed 3 rows, modifying the contents of none, and inserting 3 rows as data into the EpiMart.

The summary table at the end of the report shows totals for the amount of time each operation took and the number of rows, bytes, and metadata objects that were extracted. Using this information, you can quickly determine if a problem resides in EpiChannel, the semantic instances, or in the extraction SQL statements or system calls.

09625518.072500

Epiphany Database Extraction

Job testjob on instance epiwb. InitialLoad
(Meta DB 'chnCmd' on SQLServer nessie::epiwb_epimeta)
(Working file Job Log at
d:\metadata\logfiles\chn_1998-04-01_09-03\testjob.log)
(Log file 'FileLogging' at
d:\metadata\logfiles2\chn_1998-04-01_09-03\testjob.log)
(Log DB 'DBLog_1' on SQLServer nessie::epiwb_epimeta)

===== Preparing Job =====
All databases exist and can be opened

Checking tables ...
Verified existence of 20 tables in data store Epimart

Truncating ...
Truncated 20 tables in data store Epimart

===== Executing Job =====

-- 2.2 Processing step 'testjob' as an extractor
Testdata ==> TestDest_svr

Statement Name	Time	Metric 1	Metric 2	Metric 3
St testjob dump tran				
> dump tran on meta				
St testjob extraction stat				
<> cust real		0 St	0 Ex	0 En
<> product real		0 St	0 Ex	0 En
<> territory real		0 St	0 Ex	0 En
<> program real		0 St	0 Ex	0 En
<> appl real		0 St	0 Ex	0 En
St testjob semantic instan				
SI Product	1 Ti	0 IN	0 MO	0 PR
SI Application	1 Ti	0 IN	0 MO	0 PR
SI Customer		0 IN	0 MO	0 PR
SI Territory	1 Ti	0 IN	0 MO	0 PR
SI Program		0 IN	0 MO	0 PR

Figure 2-4 Sample Job Output (Preparation and Execution)

```
St testjob semantic instan |           |           |
...[Operations for St testjob are deleted from this sample.]

-- 3.3 Processing step AggBuilder as a system call
Statement Name              Time              Metric 1    Metric 2    Metric 3
-----
Sy AggBuilder               |          8 T1|          8 Tm|

-- 4.4 Processing step 'HotSwap' as an extractor
Testdata ==> TestDest_svr

Statement Name              Time              Metric 1    Metric 2    Metric 3
-----
St Calc Extraction Times --|           |           |           |
> Get Customer extractio|           |           |           |
> Get Product extraction|           |           |           |
> Get max of the maxes |           |           |           |
St HotSwap -- EpiCenter ind|           |           |           |
> Hotswap 2                |           |           |           |

Successfully finished body of job testjob

===== Committing Job =====

Label      EntireJob testjob AggBuilde HotSwap
-----
Time Totl|          9|          |          8|
Time Latch|          |          |          |
Time SysC |          8|          |          8|
Time Extr |          |          |          |
Time SI   |         11|         11|          |
Time AT   |         10|         10|          |
Time Elks |          5|          5|          |
Time SQLPa|          3|          |          |
Time JobR |          4|          |          |
Time Cnt R|          |          |          |
Extr Rows|          0|          0|          |
Bytes Ext|          0|          0|          |
MetaObjs#|         498|          |          |

Committed Job testjob
```

Figure 2-5 Sample Job Output (Summary)

Error Messages

An EpiChannel error message distinguishes between externally generated errors, such as database errors, and internally created messages, such as a file data store that references a nonexistent directory. Each error includes information about the database/file that caused the error, and which specific part of the code detected the error. Most errors contain information about the operations affected or aborted by the error, the metadata objects that specified these operations, and the program's plan of revised actions. Many errors also suggest corrective action.

There are specific error messages for the following:

- SELECT statements that do not match columns in the metadata. These display mappings indicate which return values were or were not accepted.
- Rows that fail to insert have their complete set of data displayed.

EpiChannel Debugging Levels

EpiChannel's verbose (**-v**) option allows you to select how much of the SQL displays on the screen during extraction. For a description of each debugging level, run EpiChannel with the **-h** or **-?** parameter. The default level shows jobs, steps, SQL statements, semantic instances, and statistics.

The lowest level shows only jobs and errors. Increasing levels show the SQL pre- and post-expansion of macros, templates, and blocks within semantic instances, or even each individual row returned from a SELECT statement. At the highest levels of verbosity, execution pauses between SQL statements and between returned rows, which allows you to trace behavior precisely.

You may even use EpiChannel interactively to alter the data fetched by a SELECT statement before the row is forwarded for insertion. See *Setting Breakpoints* below.

Note: To turn off verbosity, select -4.

Setting Breakpoints

You may set breakpoints on SQL statements that change the debug level *before* the SQL is issued. These breaks can also be associated with particular rows in the return set. For example, it is possible to set a breakpoint just before row 3021 of the *Orders table extraction* SQL statement.

You can use the EpiCenter Manager's SQL Statement dialog box to set breakpoints (see Figure 3-19, page 98). Follow these steps:

1. Select the debug level. These levels correspond to the verbosity levels on the **extract** command line (or in the Execute Job dialog box shown in Figure 3-22, page 105).
2. Select the row number at which the debug level you selected above will go into effect; this is the row number immediately *before* the SQL statement is executed. If the row number is not empty, and the level is "row pause" or higher, the pull/push loop stops fetching before inserting that particular row and waits for permission to proceed. You may alter the data during this pause.

When you set a breakpoint, it stays set for the remainder of the job. This prevents you from having to set a debug level on entire groups, but it does necessitate your setting the breakpoint back to the original value later in the job flow if you do not want the entire job debugged (and remembering to remove this breakpoint).

All SQL issued against the source, destination, or metadata databases will be logged, if specified by the debug levels.

Trial Runs

You can run EpiChannel in trial-run mode, in which SQL is not actually issued against the source or destination systems. This option is recommended for testing the structure of an extraction job.

Maximum limits can be set on the number of rows to be fetched from any SELECT statement. When used, these limits essentially cause the SQL to be checked for syntax on a small selection set.

The execution time for all SQL executed is tracked and logged.

EpiChannel Output

You must provide EpiChannel with one and only one file data store with the role of *working directory*. A working directory is a log directory that is also used as the location for temporary files associated with the job.

You may, however, have any number of file or database data stores that have the role of *log*. The locations referenced by data stores with these roles are known as log directories and log databases, depending on their data store type. You may use more than one log database and more than one log file. In this case, all logging activity is duplicated to all of the listed data stores. Having a backup of each kind of data store is recommended if you require logging information to be constantly available for the administration of your system, or if data stores could possibly go off-line or be destroyed.

You may also use multiple listings so that one log is local to the host and one is remote. For example, during installation and early testing, you might direct one database log to a database hosted by Epiphany for the monitoring of your site's activities.

Note: The EpiCenter Manager default data stores may be sufficient for your site.

Log and Working Directories

The location specified for a *log* or *working directory* is a parent directory under which a new subdirectory will be created for every job. The subdirectories are named based on a timestamp from the start of the EpiChannel run. If there is already a subdirectory of that name, then a suffix is added to make the name unique to that directory. Under the subdirectory is a file called *jobname.log* with the log and informational messages from that job.

Log Files versus Log Databases

EpiChannel can log to both files and databases. Log files provide detailed information regarding a single job and log databases provide complementary but less detailed status and performance information about all jobs. Log files help you to determine the success of a job and suggest corrective action. Log databases enable you to analyze the behavior of jobs over time in order to detect troublesome areas, or areas of degrading performance.

Note: A log database is not a log device. A log device is a construct in SQLServer that logs database transactional activity.

Monitoring Jobs

You may use the NT Performance Monitor to determine the current status of the database extraction process; for example, you can find out if the cause of a longer than normal extraction process is the result of network performance problems, or simply more rows being extracted and processed.

You may use one or more of these measures to monitor job behavior:

- The total number of jobs started. Use this to see transitional boundaries in the work being done.
- The total number of extraction nodes started. Use this to see transitional boundaries in the work being done.
- The total number of rows transferred in a pull/push operation. This value is reset with each extraction statement.

- The total number of bytes transferred in a pull/push operation. This value is reset with each extraction statement.
- The number of rows committed. This value is reset with each extraction statement.
- The rate of row transfer in a pull/push operation (rows per second).
- The rate of byte transfer in a pull/push operation (bytes per second).
- Total number of semantic instance blocks called. Use this to see transitional boundaries in the work being done.
- The rate at which semantic instance blocks are called. This shows the percentage of the clock time spent in semantic instance calls.
- The number of meta objects in the job, which is a measure of the complexity of the workflow.

You may combine these metrics with the other measures available through the Performance Monitor. For example, the bytes per second can be displayed along with the TCP packets per second, the average length of the disk queue, and SQLServer-specific measures.

Running the Performance Monitor

See *Setting Up NT Performance Monitoring for Extraction (Optional)* on page 222 for instructions on how to set up the Performance Monitor for EpiChannel usage.

After setting up the Performance Monitor, you can monitor any EpiChannel jobs that are run on this machine. Follow these steps:

1. Start the NT Performance Monitor. (It is located in the *Start\ Programs\Administrative Tools* menu on most machines.)
2. Click the ADD or plus (+) button in the Performance Monitors' window.

3. Select Epiphany Channels from the object list.
4. Select the measures you want to monitor from the counter list.

Mirroring: A and B Tables

The Epiphany system uses the metadata definitions of facts and dimensions to construct the physical EpiMart tables. Table naming conventions indicate how the table is used. For instance, if a base dimension table named Customer has been defined via EpiCenter Manager, the system constructs these tables:

- Customer_O_A
- Customer_O_B
- CustomerMap_A
- CustomerMap_B
- CustomerStage

The *_O*, *Map*, or *Stage* suffixes are needed for the normal operation of EpiMart. Note that the *Customer_O* and *Map* tables appear twice in this list: once each with a suffix of A and B. The purpose of this table “mirroring” is to allow extractions to occur without disrupting the live usage of a system.

At any time, either the letter A or B is “active,” meaning that all end-user queries will operate against tables with that suffix. This setting is determined by an option in the EpiCenter Manager’s Configuration dialog box (described in Appendix C) called *current_datamart*. If this option is set to A, then all end-user queries will be routed against tables that end in A (and all of the B tables will lie dormant). The B tables contain data that is one extraction older than the A tables.

During an extraction, the data in the active tables (A in this example) is never changed. Instead, the B tables are constructed by combining the A tables’ data with any newly extracted data from the staging tables. In other words, the dormant set of tables is rebuilt every extraction with the latest data. Not only does extraction of the base tables (those that end with *_O*) occur in the dormant set, but aggregation is also performed on the B tables. In this way, end users

can continue to query against the A tables, without knowing that extraction and aggregation are occurring simultaneously against the B tables.

If extraction finishes successfully, then the value of the configuration setting *current_datamart* is toggled to be the dormant letter (B in this case). As soon as the Epiphany Application Server (described in Chapter 5) is notified that the data extraction was successful, all end-user queries will be directed against the B tables. In this way, the A tables, which contain one less extraction's worth of data, become the dormant set. The next time EpiChannel is run, the A tables will be recreated, and the letter A will once again become active.

Note the following:

- Aggregate building is always performed against the set of tables not equal to the *current_datamart* setting.
- In order to completely re-extract an entire system, the set of tables whose suffixes equal the *current_datamart* setting must be truncated. (This, however, affects current end users of the system and will be resolved by the introduction of a new semantic type that allows complete re-extraction without truncating any EpiMart tables.)

CHAPTER THREE

EpiCenter Manager

EpiCenter Manager is Epiphany's versatile program for configuring, administering, re-configuring, and updating your database. You will use its graphical user interface to define your EpiMeta, or database table schema, as well as the jobs needed for data extraction. You will also use EpiCenter Manager to perform administrative tasks, such as restricting access to ticksheets and sending e-mail notification of job status. When your schema changes incrementally, EpiCenter Manager allows you to update it "on the fly," without rebuilding the entire EpiMart.

This chapter describes how to use EpiCenter Manager to set up your databases for optimal extraction of your source data, to define jobs, and to administer your system. The following basic concepts are also discussed:

- Uniform Treatment of Time
- Uniform Transactional Data
- Epiphany's Adaptive Architecture

Planning Your EpiCenters

EpiCenter Manager is a program for experienced database administrators who are familiar with their site's database and database needs. Because incorrectly updating schema can corrupt an existing database, it is important that you understand Epiphany system concepts and how to use

EpiCenter Manager in a way that is appropriate for your site. Before using EpiCenter Manager, be sure to read Chapters 1 and 2 of this *Guide*, which describe the Epiphany system and the database extraction process. Many of the key terms, such as EpiCenter, EpiMeta, and EpiMart are described in these chapters.

Before you configure an EpiCenter, your site needs to analyze its business model to determine:

- How many EpiCenters are appropriate.

An EpiCenter is a construct for organizing your metadata into a database called EpiMeta that defines a single EpiMart database. EpiMart is where the data extracted from your source systems resides. If your system has adequate memory and storage (see Appendix A, *Installation*), and your business model warrants it, you may set up multiple EpiCenters. For example, you may have one for development and one for production.

- The number of constellations within each EpiCenter.

This includes the fact tables and dimension tables (and aggregates) that should be logically organized into a constellation. Although every EpiCenter has at least one constellation, some EpiCenters will have multiple constellations. The number depends on the site's business model; for example, one for each business process.

- The extraction process for your site.

Analyze the kinds of jobs and job steps needed; for example, you may create job flow chart(s). After you design the system at this higher level, you will be better able to—

- Identify the extractors (SQL statements and semantic instances) needed to extract information from the source system(s) and write these extractors.
- Define external tables.
- Identify requisite system-call commands.

Getting Started

Follow these steps to start EpiCenter Manager and register a database server:

1. Complete the installation as described in Appendix A, *Installation*.

The installation procedure includes creating new, empty databases on the server for the EpiMeta and EpiMart databases and installing the Epiphany software. Although you create two databases, the EpiMeta is the one you work with using EpiCenter Manager. The EpiMeta defines the database tables for your data warehouse (the EpiMart).

2. Start EpiCenter Manager.

During installation this program executable (**EpiMgr.exe**) is placed in the Epiphany directory by default (*C:\Program Files\Epiphany*) and can be selected from the *Start* menu:

Programs\Epiphany\instance_name\EpiCenter Manager.

The EpiCenter Enterprise Manager window is displayed.

At the top level is the Servers icon. Follow these steps to register your server:

1. Choose Register from the Server menu. The Server Properties dialog box (Figure 3-1, page 60) is displayed in the window.
2. Select the Server Type from the drop-down list and enter the database server's name, and the username and password for this server.

Your database server machine icon and the EpiCenters folder (or directory) appear in the hierarchical tree structure (see Figure 3-2, page 61). This server icon represents the server where the EpiMart and EpiMeta databases for this EpiCenter reside.

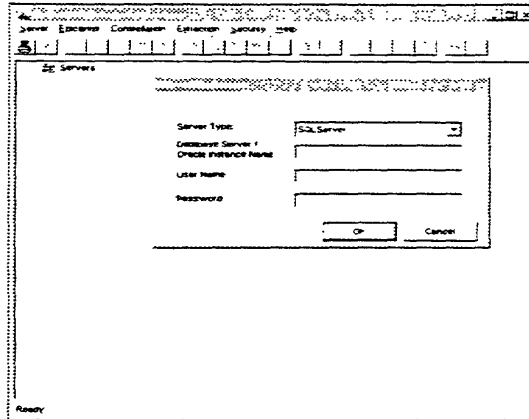


Figure 3-1 Registering the Database Server

3. Click Cancel to close the empty Choose EpiCenter dialog box.
Later you may use this dialog box to select and open an already initialized EpiCenter as described in *Working with an Existing EpiCenter*, page 66.
When you register a server for the first time, after connecting to that machine, EpiCenter Manager displays the Register EpiCenter dialog box. Close this dialog to continue with the initialization.

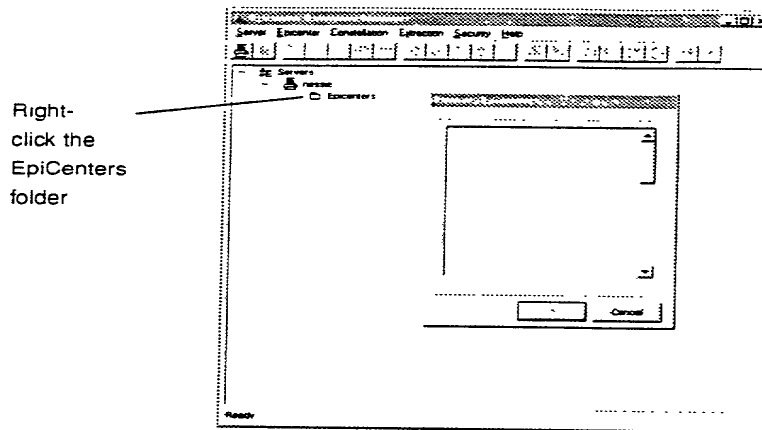


Figure 3-2 EpiCenter Enterprise Manager Window

Follow these steps to initialize your EpiCenter:

1. Right-click the EpiCenters folder in the EpiCenter Manager Enterprise window (see Figure 3-2) and select Initialize EpiCenter from the pop-up menu. The Initialize EpiCenter dialog box is displayed (see Figure 3-3, page 62).
2. In the EpiMeta Name drop-down list, select the name of the new, empty database for the metadata that you created during installation (as described in Appendix A, *Installation*).
3. In the EpiMart Name drop-down list, select the name of the new, empty database for your customer data that you created during installation.

4. Click Build to start building a generic EpiMeta database of the size you specified. The status of the build is displayed in the lower part of the (now expanded) Initialize EpiCenter dialog box.

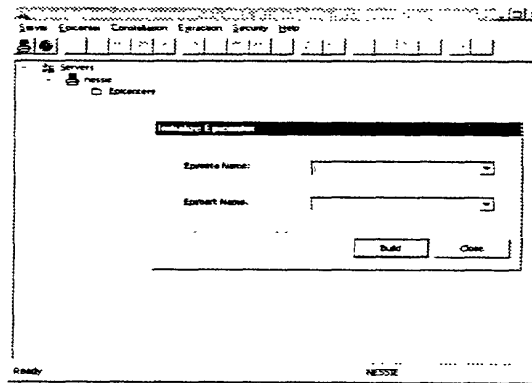


Figure 3-3 Initializing Your EpiCenter

The new EpiCenter icon is added to the directory tree with folders for Base Dimensions, Constellations, Extraction, and Security (see Figure 3-4). Note that the icons in the window are arranged in an Explorer-like hierarchical tree.

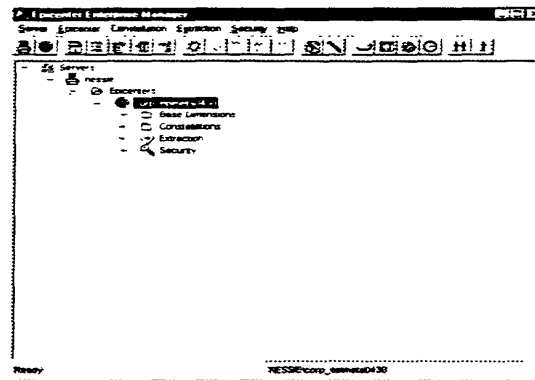


Figure 3-4 EpiCenter Enterprise Manager Window

Next, the Date Dimension dialog box is displayed. You will need to populate the actual, physical date dimension table. This is a special base dimension table supplied by Epiphany that is used for storing all attributes related to time. All fact tables in an EpiCenter receive a foreign key (called *date_key*) to this table.

Note: If you receive the message: *Cannot open the datamart in single-user mode*, close any other applications that are currently using this database, including the SQLServer Manager in which the database was created. For more information about this error message, see page 68.

To populate the date dimension table:

1. Choose Populate Date Dimension from the EpiCenter menu.
2. In the Date Dimension dialog box (Figure 3-5), select the values for the beginning and ending year of the EpiMart and your calendar type and start day. The date range of the EpiMart should be at least as large as any

dates that are found in the data you will be extracting. These values are defined in the Configuration dialog box (choose Configuration from the EpiCenter menu to open it).

In order for users to get the best results when forecasting trends using Relevance, you need to build the date dimension as far into the future as you plan to forecast. (Currently, the maximum prediction is three years past the last date that has recorded data.) If the date dimension is not built out far enough, the user will receive columns with names such as *Dec 1999, Second Next, Third Next* instead of *Dec 1999, Dec 2000, Dec 2001*.

Note: Quarters 4-4-5 represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.

3. Click Build. The EpiMeta database will be populated with a base list of dates.

For descriptions of these date fields, see *Configuration*, page 243 in Appendix C and Appendix D, *Date Dimension Fields*.

Figure 3-5 Date Dimension Dialog Box

A discussion of how the Epiphany system treats time follows. Please read this section and read the introductory material in *The EpiCenter Manager Window* and *Working with EpiCenter Manager*, page 67 before beginning to set up your EpiCenter as described in *Setting Up an EpiCenter*, page 69.

Basic Concept: The Uniform Treatment of Time

In many Enterprise Relationship Management (ERM) applications, the time on which a fact occurs is a crucial dimension. It can be complicated, however, to create the semantics of calendars and other date attributes for a new data warehouse. Epiphany provides a time dimension via the special table *Date_O*, along with EpiCenter Manager, for populating the date dimension with the parameters that make sense for each implementation. Each row in this table represents a single day and the attributes associated with that day.

Every fact table in EpiMart automatically contains a foreign key to this special date dimension. Many of the fact semantic templates (the generic programs that perform business transformations on extracted data) depend on the presence of this field. By joining the fact table to the date dimension, Epiphany's ERM applications are able to answer questions of the form:

- Which facts occurred this quarter?
- What is my weekly backlog for the year?
- What products were bought last month?

The queries that are constructed for these purposes do not perform date arithmetic, such as:

```
SELECT * FROM MyFact
WHERE date_key > '6/1/1998' and date_key < '7/1/1998'
```

Queries of the above form become unwieldy because of the non-uniformity of the calendar with respect to the number of days in the month, leap years, weekly overlaps with months, and so on. Instead, determining which days belong to which weeks, months, quarters, or years is done once when the date dimension is populated. This table is then fully enumerated with all values of interest for the EpiCenter, allowing queries of the form:

```
SELECT * FROM MyFact, Date_0 WHERE MyFact.date_key =Date_0.date_key
AND cq_and_cy_name = 'Jun 1998'
```

Queries of this form can easily take advantage of RDBMS indexes.

Epiphany allows for specification of Calendar fiscal quarters, as well as 13-week manufacturing calendars (4-4-5 calendars), in which a quarter always starts on the same day of the week. The date dimension can also be configured to specify which weekdays start and end a business week (for example, Sunday through Saturday versus Monday through Sunday).

Working with an Existing EpiCenter

If an EpiCenter has already been initialized, you may register it as follows:

1. In the EpiCenter Enterprise Manager window, right-click the EpiCenters folder icon and select Register from the pop-up menu.
2. In the Choose EpiCenter dialog box, select the EpiMeta for the EpiCenter.

The EpiCenter's hierarchical tree is displayed in the EpiCenter Manager Enterprise window. You may modify this EpiCenter as appropriate for your site by following the instructions given in this chapter for those items you wish to change.

You can unregister an EpiCenter by right-clicking the EpiCenters folder icon (see Figure 3-2, page 61) and selecting Unregister from the pop-up menu. The EpiCenter's hierarchical tree is removed from the window.

The EpiCenter Manager Window

Similar to other Windows applications, you may invoke EpiCenter Manager commands by using the main menu, toolbar icons, or by right-clicking icons in the window and selecting commands from a pop-up menu. Before you start configuring your EpiCenter, familiarize yourself with the main menu commands and the toolbar icons. The main menu commands are organized by the main EpiCenter Manager functions: Server, EpiCenter, Constellation, Extraction, Security, and Help. These menus are contextual and their commands are accessible when you are working within their function. For example, the Constellation commands are available when you working with a Constellation, but may be unavailable in other contexts.

EpiCenter toolbar icons provide access to special functions such as truncating tables, or displaying dialog boxes you use on a regular basis, such as the new Job dialog box and dialog boxes for defining facts, dimensions, external tables, and users and groups. A toolbar icon must be activated (highlighted) for you to use it. To activate a toolbar icon, select a related icon in the directory tree.

Working with EpiCenter Manager

The EpiCenter Manager window is organized in a hierarchical tree structure in which you right-click a plus or minus sign to expand or collapse a directory, as in Windows Explorer. In general, you will work down the tree structure in the window, right-clicking folders or icons to display a contextual pop-up menu, or you may prefer to use the equivalent main menu or toolbar commands. The instructions in this chapter use both ways interchangeably.

Double-clicking an empty folder icon opens a dialog box in which you can define a new item, such as a new constellation. If this item has already been created, double-clicking expands or collapses the directory tree. Double-clicking an existing icon, such as the Default Job icon, opens its dialog box.

The following applies to all of your work with EpiCenter Manager:

- Descriptions in dialog boxes

The Epiphany system does not utilize the descriptions in the Description text area of the EpiCenter Manager dialog boxes. This description is for your documentation purposes only.

- Updating and refreshing

You may click the Update button in a dialog box to change the EpiMeta description of an existing row of data in the database “on the fly.”

If multiple users are running EpiCenter Manager, metadata changes made by one user are visible to others only after that user issues the Refresh command.

- Unregistering and deleting items

You can unregister servers and EpiCenters by right-clicking their icons and selecting Unregister from the pop-up menu. You can delete an item in the EpiCenter Enterprise Manager window, such as a base dimension or constellation, by right-clicking its icon and selecting Delete, or by selecting it and pressing the Delete key.

- An explanation of the error message: *Cannot open the datamart in single-user mode* follows:

Whenever the **EpiMgr** program operates on EpiMart (adding tables, populating the date dimension, and so forth), it attempts to put the database in single-user mode. This is to make sure that no one is running a query when the operation is being performed. This error message results if another connection is open to the same database. To solve this problem, close any other applications that are currently using this database, including the SQLServer Manager in which the database was created.

Setting Up an EpiCenter

The purpose of setting up an EpiCenter is to define your site's metadata, or schema. After defining it, you will use the Generate Schema command in the EpiCenter menu to write these definitions to the EpiMart database. The EpiMeta database points to your EpiMart (where your extracted data is stored).

The instructions in the rest of this chapter describe how to set up a single EpiCenter with one constellation. Some EpiCenters, however, will have more than one constellation. To create additional ones, simply repeat the procedure you used to create your first constellation.

Configuration

The Configuration dialog box contains general settings, lists transaction types, and defines how measure units are displayed. (Choose Configuration from the EpiCenter menu to open this dialog box.) Transaction types distinguish rows with different interpretations in the same fact table. For instance, an Order fact table might contain both a BOOK and SHIP transaction type, differentiated by the value of the column *transtype_key*. Other than your mail password, the information in this dialog box should need little, if any modification. (Most of these settings can be set using other EpiCenter Manager dialog boxes.) See Appendix C for more information.

Base Dimension Tables

The base dimension tables are physical dimension tables that can be used multiple times within a constellation or across constellations. All EpiCenters are created with two default base dimension tables: Date and Transtype.

A base dimension table consists of dimension columns. These are columns in a physical dimension base table that hold attributes extracted from the source system.

Use the Base Dimension dialog box to—

- define the EpiCenter's base dimension tables, such as Product, CostCenter, Account, Subaccount, or Project.

- configure the dimension aggregates used for aggregate building.

To define a new base dimension table:

Right-click the Base Dimensions folder and select New Base Dimension from the pop-up menu. The Base Dimension dialog box (Figure 3-6) is displayed.

This dialog box has three tabs: General, Column Sets, and Built Aggregates.

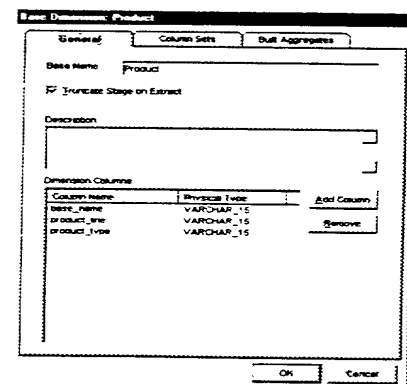


Figure 3-6 Base Dimension Dialog Box: General Tab

Use the General tab to define the base dimension table:

1. Enter the name of the base dimension table, such as *Product*. This is the name of the physical table in the database; underscore zero (_O) will be added to this name to form the base table.
2. By default, the data in the base dimension staging tables is deleted (truncated) after extraction.

In most cases, you will want to truncate staging tables. You may, however, want to de-select Truncate Stage on Extract when an execute-only SQL statement truncates the staging tables based on criteria you specify for the extraction process. For example, this SQL could delete all records whose foreign keys matched, or all records over seven-days-old.

Alternatively, you may use a single dialog box to define all tables within a constellation that will be truncated prior to extraction (see *Truncating Tables*, page 107).

3. Click Add Column to enter one of the table's column names.
4. In the Dimension Column dialog box, enter a name and a description for the column.
5. Select the physical type from the drop-down list. The physical type you select is replaced by its associated database type. The translation of physical type to database type depends on your RDBMS platform. See Appendix E, *Physical Type Values*, for descriptions of these physical types.
6. Enter a default value for the column.

By default, each dimension column assumes the literal value UNKNOWN for each of its textual attributes. Leave this field blank to accept the default. To override this value so that end-user queries will return a different value, enter a literal string (without quotes). Your string will be used instead of UNKNOWN. See *The UNKNOWN Dimension Row*, page 34.

Defining Dimension Aggregates

A dimension aggregate represents a dimension table in which one or more of the columns have been removed, and the rows have been collapsed onto one another to remove duplicates. (In the example given in *Aggregate Building*, page 28, if Customer Name is removed from the base Customer table, only two distinct Regions are found in the table, so only two rows are needed in the Region aggregate table.)

Use the Columns Sets tab of the Base Dimension dialog box (Figure 3-7, page 73) to define which aggregates will be built the next time that the Aggbuilder program (**agg.exe**) is executed.

Column sets are subsets of the full list of dimension columns for the associated dimension base table. Each column set represents a *potential* aggregate to be built: the Aggbuilder program does not build all defined columns sets, it builds only those columns sets that are included in fact aggregates (see *Aggregation and Aggregate Grouping*, page 81).

To define dimension aggregates, follow these steps:

Note: The data you enter in the Column Sets tab is used by the Aggregate Building process the next time **agg.exe** is run. Your changes do *not* affect the current EpiMart tables.

1. Enter the name of the column set in the text box, and click New.
This name is a logical identifier only, and is not used during the building of the Actual table in the EpiMart. Instead, EpiCenter Manager will append unique numbers after the base dimension table name.
2. Select or de-select Default.
Selecting Default for this column set affects the behavior of the default aggregate group that exists in each constellation (see *The Default Aggregate Group*, page 84).
3. Add column names for this set by clicking New. Select columns from the list of columns for this base dimension table.

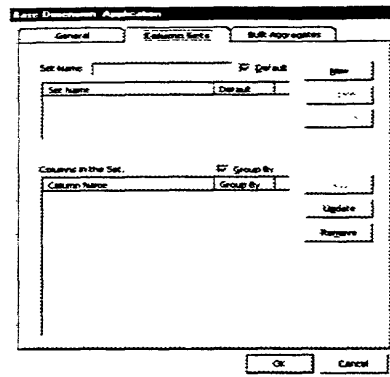


Figure 3-7 Base Dimension Dialog Box: Column Sets Tab

4. Select or de-select Group By.

When including a column in a set, you may specify whether that column is to be used in the GROUP BY clause of the SQL statement that generates the aggregate. Normally, you should select all Group By flags.

Most DBMS's, however, have a limitation of 16 columns in a GROUP BY clause. If a column's value is determined by another column already flagged as Group By, then the former column can be left out of the GROUP BY clause without affecting the result set.

For example, consider a dimension called Product which has, among other things, both a category code and a category name. If each code has a unique associated name, then the code and name should be included in the same set because any aggregation on one is an aggregation on the other. Because the two columns change together, there is no point to group by *category_number* and *category_name* because either group by produces the same results. One of these columns can be left out of the group-by list.

5. Add additional (multiple) column sets by repeating the above steps.

Multiple dimension column sets are used when a base dimension table with many columns can be aggregated in different ways. Generally, the fewer columns included within a column set, the smaller the resulting dimension aggregate, and the faster queries that use fact aggregates joined to that dimension aggregate will respond.

For instance, if a customer dimension has several fields related to customer geography (City, State, Zip, and so forth) and several other fields related to demographics (age, marital status, years of education), then EpiCenter Manager can choose two or more dimension column sets (which can overlap) for maximum coverage of queries. At query time, Epiphany's aggregate navigation mechanism will choose the smallest aggregate that satisfies the user's request.

Thus it is beneficial at query time to have built many small, highly specialized aggregates. The penalty for building many dimension aggregates is that it takes longer to build aggregates, and requires more disk space.

Note: A small change in the number of dimension aggregates can cause a large change in the number of fact aggregates. The Aggbuilder program builds only those column sets included in fact aggregates (see *Aggregation and Aggregate Grouping*, page 81).

Dimension Aggregate Browsing

While configuring your Epiphany system, you may wish to browse the list of aggregates that Aggbuilder has built (according to the instructions in the Aggregate Group dialog box described in *Aggregation and Aggregate Grouping*, page 81). You can use the Built Aggregates tab of the Base Dimension dialog box to browse the aggregates for either the A or B set of tables.

EpiCenter Manager uses a mirroring structure in which all EpiMart tables are in the database twice, once for each of the _A and _B tables. Only one set of tables, however, is active at a given time. See *Mirroring: A and B Tables*, page 55 for more information.

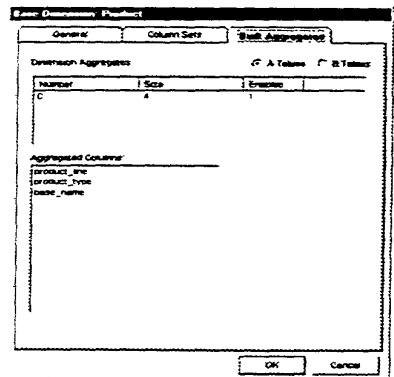


Figure 3-8 Base Dimension Dialog Box: Built Aggregates Tab

Setting Up a Constellation

Within an EpiCenter, constellations serve to group fact tables that have similar dimensions. Instructions are given here for setting up a single constellation. To set up another constellation, repeat the entire series of steps in this section.

To set up a constellation:

1. Choose New Constellation from the EpiCenter menu.
2. Enter the new constellation name and any description.

The New Constellation icon appears in the Constellation folder. It has these sub-folders arranged in alphabetical order: Aggregates, Degenerate Dimensions, Dimension Roles, Facts, Measures, and Ticksheets.

Although you can usually proceed down the tree as you define your EpiCenter, you need to define dimensions and facts before aggregates. Follow the sequence given below.

Defining Dimension Roles

A base dimension table may have a superset of data that can be used for different purposes. For example, a Customer base dimension table may contain Bill-to and Ship-to data. Dimension roles enable base dimension tables to be used more than once in a constellation.

A dimension role indicates the single usage (such as Bill-to data only) of a base dimension table by all of the fact tables within a constellation. A row in a fact table may have several foreign keys that point to the same base dimension table, but the role usage may differ (for example, some relate to Bill-to and others to Ship-to data). The dimension role maps the fact foreign keys to the correct base dimension table.

To define roles for this dimension table:

1. Right-click the Dimension Role folder and select New Dimension Role. The Dimension Role dialog box is displayed.
2. Select the base dimension name from the drop-down list.
3. Enter the Dimension Role Name and a description, and click OK. The dimension role icon is added to the tree.
4. Repeat the above steps for any other base dimension tables that have multiple roles.

Every dimension role must have a parent base dimension table. If there is no base dimension for this role, click New in the Dimension Role dialog box. Use the Base Dimension dialog box that displays to create a new base dimension.

Degenerate Dimensions

Certain numbers in fact tables such as order, invoice, and bill of lading numbers have foreign keys with no corresponding dimension table. The only data of importance is the number itself, all of the other information of value has been extracted into other dimension tables. These numbers are called degenerate dimensions. A degenerate dimension is part of a constellation, and all facts in the constellation inherit it. If you use degenerate dimensions, you need to inform the system of their existence.

To define a degenerate dimension:

Choose New Degenerate Dimension from the Constellation menu. Enter the degenerate dimension name and a description in the Degenerate Dimension dialog box.

Defining Fact Tables

A fact table is a physical table that holds numeric data in its columns. It also represents the intersection of a series of dimension keys. A fact table consists of dimension role foreign keys, degenerate dimension keys, and fact columns.

To define a fact table:

1. Right-click the Facts folder icon, and select New Fact. The Fact Table dialog box (Figure 3-9) is displayed. This dialog box has three tabs: General, Custom Index, and Built Aggregates.
2. In the General tab, enter the fact table's name and text that describes this fact table.
3. By default, the data in the fact staging tables is deleted (truncated) after extraction. In most cases, you will accept the default.
4. By default, the *Build aggregates for this Fact* check box is selected. Deselect it to turn off aggregate building.

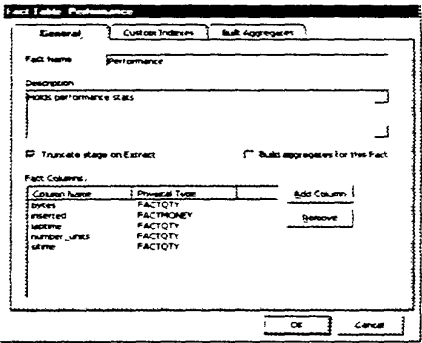


Figure 3-9 Fact Table Dialog Box

Defining Fact Columns

Fact columns contain the actual customer numeric data; such as, *net_price* or *number_units*. You will use the General tab of the Fact Table dialog box to define a fact column:

- 1. Click Add Column. The Fact Column dialog box (Figure 3-10) is displayed.

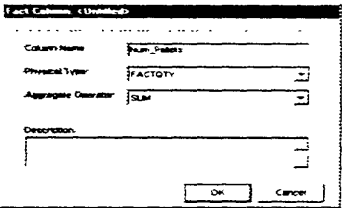


Figure 3-10 Fact Column Dialog Box

2. Enter the column name and select the physical type from the drop-down list. The physical type you select is replaced by its associated database type. The translation of physical type to database type depends on your RDBMS platform. See Appendix E for descriptions of these physical types.
3. Select the aggregate operator from the drop-down list. Aggregate operators determine how the Aggbuilder program calculates the facts. (For this release, SUM is the only option.)
4. Enter a description, and click OK to return to the Fact Table dialog box. The new fact column appears in the listbox.
5. Repeat the above steps to define another column.

To remove a column name, select it and click Remove.

Custom Indexes

Custom indexes can dramatically improve query performance. See *Custom Fact Indexing*, page 33 for background information.

Use the Custom Indexes tab of the Fact Table dialog box (Figure 3-11) to define indexes.

To define a custom index:

1. Enter the name of a custom Index in the text box and click New. The index name is added to the list.
2. For *Dimensions in the index*, select the index in the list and click Add Dimensions. The Choose Dimension dialog box is displayed.
3. Select one of the dimension roles from the list.
4. Click OK to add this dimension to the list.

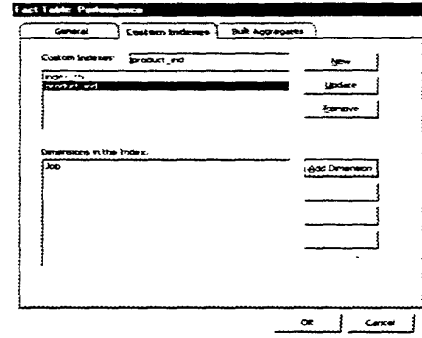


Figure 3-11 Fact Table Dialog Box: Custom Indexes Tab

Fact Aggregate Browsing

While configuring your Epiphany system, you may wish to browse the list of built fact aggregates to determine which tables are actually available for the query machinery. (See *Aggregation and Aggregate Grouping*, page 81 for more information.)

Use the Built Aggregates tab of the Fact Table dialog box (Figure 3-12) for this purpose. As with dimension aggregate browsing, you can browse either the A or B set of tables, as appropriate. Follow these steps:

1. Select a dimension role from the Role panel and then select a filter from drop-down list. (Make one selection per dimension role.) The filter options are described in Table 3-1 on page 84.

Filtering specifies that a dimension role—

- should not be included in the aggregate.
- should be included as the base table.

- should be included as a specific dimension aggregate (corresponding to a dimension column set).
2. Fact aggregates appear in the Fact Aggregates listbox.
- Clicking a row in this listbox displays the following in the Aggregate Contents panel of the tab: the dimension roles that are included for this fact aggregate, and the dimension aggregates (or the base table) that the fact aggregate joins to.

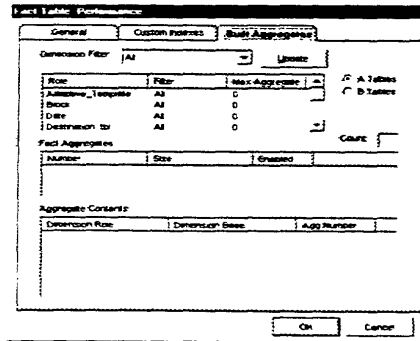


Figure 3-12 Fact Table Dialog Box: Built Aggregates Tab

Aggregation and Aggregate Grouping

Aggregates are facts that the system will pre-calculate for a group of dimensions. (These dimensional groupings are determined by the groups you set up for a base dimension table.) An *aggregate group* is a set of instructions (a metadata definition) that tells Aggbuilder which aggregates to build on one or more fact tables. The fact aggregates that are actually built are determined by a combinatorial expansion of the instructions in the aggregate group.

Unlike dimension aggregates, fact aggregates are not fully enumerated through the use of EpiCenter Manager. The reason for this is simple: while base dimensions typically have only a few aggregates, facts usually have many.

EpiCenter Manager simplifies aggregation by providing aggregate groups that you define using the Aggregate Group dialog box (Figure 3-13, page 85 shows this dialog box for the Default Aggregate Group). For each aggregate group, you will specify the facts in the constellation that share it. (An aggregate group applies to a single constellation.) If two or more facts share an aggregate group, then the Aggbuilder program will follow the same set of instructions for each of the facts.

To open an existing Aggregate Group dialog box, double-click it. (To open a new Aggregate Group dialog box, expand the Constellation tree and right-click the Aggregates folder. Select New Aggregate from the pop-up menu.)

Use the General tab of the Aggregate Group dialog box to define an aggregate group:

1. Enter an aggregate group name (and description).
2. Select Enabled for Aggbuilder for the Aggbuilder program to use this grouping.
3. Select the facts that share this aggregate. Click Add Fact and select facts from the Choose dialog box.

Use the Definition tab of the Aggregate Group dialog box to configure the dimensionality:

1. Click the Add Dim button.
2. Select the dimension roles in this constellation (to be considered for these aggregates) from the Choose dialog box.

If a dimension role is not included in the aggregate group, then *all* of the aggregates that result from this aggregate group will not include that dimension role.

3. Select a dimension role (in the Dimension Settings listbox) and click Add Set to configure the Other Column Sets aggregate type for this dimension (see Table 3-1).

Dimension aggregates are built during the aggregation process. Each column set for a base dimension will become a dimension aggregate, as long as that column set is included in some aggregate group in the Other Column Sets list.

When a dimension role is included in an aggregate group, you need to specify the possible ways that the dimension role can be built into the aggregates defined by this group. To illustrate this point, assume that you define an aggregate group that consists of the following dimension roles:

Dimension Role	Number of Possibilities
Customer	3 (Not included, Set 1, Set 2)
Product	3 (Set 1, Set 2, Set 3)
SalesPerson	2 (Not included, Base)
Date	5 (Not included, Sets 1-4)

This aggregate produces $3 \cdot 3 \cdot 2 \cdot 5$, which equals 90 aggregates for each fact table included in the aggregate group. Aggbuilder will assign a unique number to each of these aggregates, and the resulting table that is built in EpiMart will be *fact table name_#*.

The following table summarizes the choices for inclusion of a dimension role in an aggregate group.

Table 3-1 Dimension Roles for Aggregation

Inclusion	Description
All	The dimension role can either be excluded from an aggregate, have its base table included, or have one of the Other Column Sets included.
NoneOrOthers	The dimension role can either be excluded from an aggregate, or have one of the Other Column Sets included.
BaseOnly	The dimension role will always be included in all aggregates as the Base Dimension Table.
OthersOnly	The dimension role will be included only as one of the Other Column Sets.
BaseOrOthers	The dimension role can either have its base table included, or have one of the Other Column Sets included.
NoneOrBase	The dimension role can either be excluded in an aggregate, or have its base table included.

The Default Aggregate Group

Each constellation has a default aggregate group that has special semantics to make the configuration of aggregates simpler. You cannot delete this group, although you may disable it.

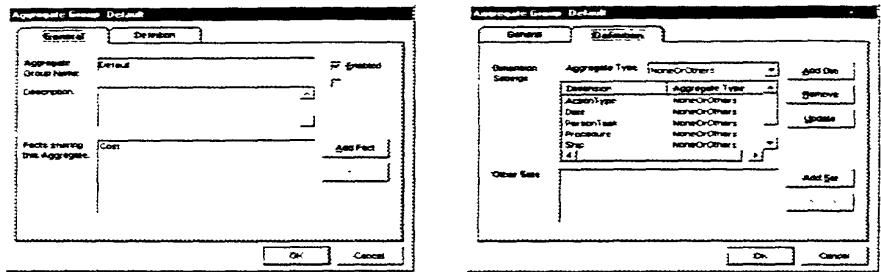


Figure 3-13 Default Aggregate Group Dialog Box

The date dimension is automatically included in this group, since many end-user queries involve time. Date is included as a NoneOrOthers dimension role, and the Other Column Sets listbox is determined by which column sets of the Date Dimension dialog box are set to Default.

When new dimension roles are added to the constellation, the default aggregate group is automatically modified to include those roles in a NoneOrOthers manner. The reason for excluding the base tables from the default aggregate group (by default) is that base dimension tables are typically much larger than dimension aggregates, so building fact aggregates that include one or more base dimension tables is expensive in terms of time and disk space.

When this dimension role is added to the default aggregate group, the Other Column Sets listbox is populated automatically with each column set that is declared as Default for the base dimension table to which the dimension role joins. You may assign the default flag of a column set to 1, when it makes sense for that set to be included by default in many fact aggregates.

In all other respects, the Default Aggregate Group is identical to any other aggregate group.

This completes the section on setting up a constellation. To set up another constellation, return to *Setting Up a Constellation*, page 75.

Measures and Ticksheets

Note: Measures and ticksheets will not exist until after you have configured your EpiCenter and have specifically created them using the Web Builder program, which is discussed in Chapter 4.

A measure is a single business calculation, and a ticksheet is a user interface form in Epiphany available to end users for constructing queries. Measures and ticksheets are integral components of the Clarity and Relevance applications.

If measures or ticksheets exist for this constellation, right-clicking the plus sign next to a Measures or Ticksheets folder displays their names. Ticksheets have associated dialog boxes. Double-clicking a ticksheet dialog box (Figure 3-14) enables you to view a list of Saved Queries (reports) for a ticksheet and to view the assigned permissions for these queries.

The Saved Queries panel of the dialog box lists all Saved Queries for this ticksheet. You may select a saved query and click Remove to delete it. The entire list can be deleted; for example, you may want to remove “orphaned” queries. These are queries that cannot be accessed by anyone via Epiphany’s front-end Application Server, and may occur when a user is deleted.

Select the Only Show Orphans option in the dialog box to display orphaned queries for deletion.

The permissions for the selected saved query are shown in the lower panel of the tab. You may use the Permissions tab to delete (but not add) individual permissions for a saved query. Select the owner and click Remove.

TicketSheet Reason

Name:

TicketSheet Type:

Deleted:

Saved Quotient: ☐ ☐

Quotient Name:

Parameters	Values
Quotient	10

Figure 3-14 TicketSheet Dialog Box

See *Security* on page 109 for a discussion of permissions.

Basic Concept: Uniform Transactional Data

A transaction represents an event in time. All data in an EpiMart fact table is stored in transactional format. For certain business entities, this format makes intuitive sense, such as in the case of an invoice in which a record in EpiMart represents the shipment of a product to a customer at a certain point in time. For the most part, this event cannot be changed; it simply happened and is stored as such in an EpiMart fact table.

Other business entities are not so easily made into transactions. When a customer calls to order a product, he might choose to order 10 units. In the Epiphany system, this fact is entered as a transaction with quantity 10 for Product P1 to Customer C1.

Order Fact

Customer	Product	Date	Quantity
C1	P1	6/1/1998	10

However, if the customer calls back the next day and changes the order to 15 units, then instead of entering a separate transaction of 15, Epiphany's extraction machinery enters the *difference* of 5 as a new transaction on that second day.

Customer	Product	Date	Quantity
C1	P1	6/1/1998	10
C1	P1	6/2/1998	5

Similarly, in the case of Inventory, Epiphany tracks changes in inventory as transactions instead of restating the reported inventory from the outside world. For example, if Customer C1 *reports* inventory for Product P1 as:

Reported Inventory in Source System

Date	Quantity On Hand
6/1/1998	10
6/8/1998	12
6/15/1998	5

In Epiphany's fact table the same information is represented transactionally as:

Inventory Fact

Customer	Product	Date	Change In Inventory
C1	P1	6/1/1998	10
C1	P1	6/8/1998	2
C1	P1	6/15/1998	-7

Of course, an end user who asks for a report of inventory by week wants to see the data reported in Clarity as it is reported by the source system. To accomplish this, use Web Builder to define a measure that has a backlog type that makes this measure an accumulator over time. This causes the transactions to be reassembled back into the reported format.

Why go to the trouble of disassembling orders and inventories into transactions, only to reassemble the previous format at output time? The reason is that transactions are the most additive way to store data, which means that transactions can be recombined along arbitrary dimensional boundaries.

The way in which Inventory data is reported above can easily answer queries of the form "Inventory by *week*." This data, however, cannot easily be used to answer queries by month or year without some external knowledge of which week is the end of the month. Today's RDBMS engines cannot efficiently handle these types of queries without Epiphany's transactional storage. Note that the inventory for the month of June can be calculated by adding up all the transactions that occur between 6/1 and 6/30, yielding an ending inventory of $10 + 2 - 7 = 5$.

Important: All source system data must be made into transactions. Keep this in mind when you author an Epiphany extractor or choose a semantic type.

Extraction

After you have configured your EpiCenter, you may use the Extraction folder to set up the extraction process described in Chapter 2. The Extraction folder consists of the sub-folders Data Stores, External Tables, Extractors, and Jobs, each of which is discussed below.

Data Stores

Use the EpiCenter Manager's Data Store dialog box to create as many data stores as are appropriate for your site. Each extractor has a single input and output data store. Because jobs consist of multiple extractors, a job has multiple input and output data stores. For example, one job might extract data from two source systems into one EpiMart, and another job might extract data from a source system into external tables and from external tables into staging tables. In the second job, the external tables serve as both input and output data stores to different extractors.

The Data Store Dialog Box

Each data store has a Data Store dialog box (see Figure 3-15), and this dialog box has two tabs: General and Properties.

The General tab has two panels: Data Store Type and Data Flow. The Data Store Type panel of the General tab is where you enter the name and description for the data store and select its type: Microsoft SQLServer, Oracle, Generic ODBC Data Source, or File.

The Data Flow panel of the General tab has these options:

- *Allow Use as Input Data Store* allows the data store selected in the Extractor dialog box (Figure 3-17, page 95) to be used as the input of an extractor.

The source systems listed in the Source System drop-down list serve to distinguish source system identifier keys that may clash between different database systems. For example, if an account has an ERP and an

SFA system, each of which has a Customer table, both may contain a customer number 100. To distinguish between these two records, two different source system identifiers must be used.

Important: Fact rows will join only with dimension rows that have the same source system identifier. For this reason, unless the site actually uses two or more logical source systems, select Datamart Source from the Source System drop-down list (the default).

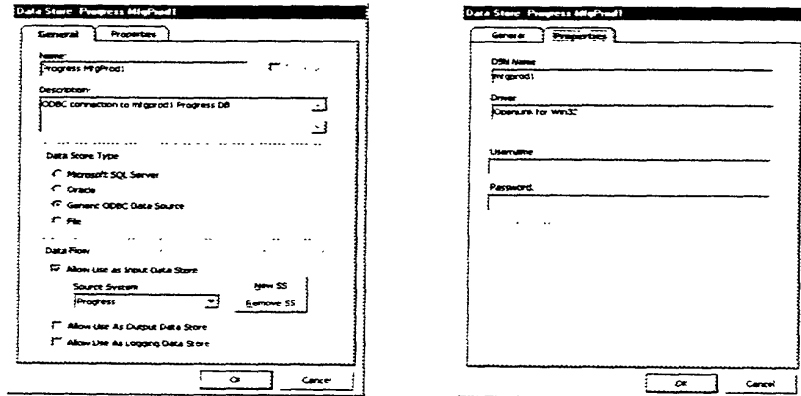


Figure 3-15 Data Store Dialog Box: General and Properties Tabs

- *Allow Use as Output Data Store* allows the data store selected in the Extractor dialog box (Figure 3-17, page 95) to be used as the output of an extractor.

De-selecting this option ensures that one does not accidentally write data to a database that is read-only; the data store name does not appear in the list of available output stores for extractors.

- ***Allow Use as Logging Data Store***

Epiphany has special metadata tables for the purpose of logging its activity. Normally, this logging is written to the EpiMeta database.

Logging to the EpiMeta, however, may increase its size significantly, so you may prefer to log to another data store. You can use the *logging_mssql.sql* script (included with the Epiphany software) to create a new logging database. If you run this script on another database, then a new data store can be used to log extractions.

Note: The *Allow Use as Logging Data Store* option must be selected in the Data Store dialog box. The purpose of this option is to ensure that logging is directed to the proper logging data store.

The Properties tab has different fields, depending on the data store type. Use the Properties tab to enter these fields:

- For SQLServer, enter the server's name, database name, username, and server's password and version.
- For Oracle, enter the SQL Net name, username, the server's password, and the version number (Oracle 7 or Oracle 8).
- For a generic ODBC data store, enter the DSN name and ODBC driver name, and server administrator's username and password. (You must also set up a DSN for the data store system using the Data Source Administrator in the Windows NT Control Panel.)
- For the *JobFileLog*, enter its file directory path.

Modifying the Default Data Stores

You will need to modify the default data stores, which are *EpiMart*, *JobFileLog*, and *LoggingDB*. Double-click their folders to open them and fill out the dialog boxes as follows:

- *EpiMart* specifies your EpiMart as a data store.

The only value that you can configure is the name of your EpiMart. Use the Properties tab of the EpiMart Data Store dialog box to enter this database name, for example *Corp_EpiMart*.

- *JobFileLog* specifies the data store for EpiChannel job logs.

In the General tab, the settings are correct for the default usage. The Data Source Type is a file, and the Data Flow is set to *Allow Use as Logging Data Store*.

Click the Properties tab, and change the directory for the *JobFileLog* from CHANGE ON INSTALL to the correct directory name. A valid directory name is required for a successful extraction. Leave the file name blank.

- *LoggingDB* specifies the data store for the EpiChannel logs.

In the General tab, the Data Source Type is your server, and Data Flow specifies: *Allow Use as Input Data Store* and *Allow Use as Logging Data Store*.

You can accept the default values for the log database's file name and directory path, unless you have located the log in a different database.

Note: \$\$DEFAULT refers to the current EpiMeta.

External Tables

Extraction statements are sometimes directed to load external tables that serve as intermediary tables for multi-staged extraction. Epiphany supplies one external table called *last_extract_date*. Use of this table is recommended, but optional. See *External Tables*, page 24.

To define external tables and their columns:

1. Choose New External Table from the Extraction menu.
2. Enter the name and any description for the table in the External Tables dialog box (see Figure 3-16).
3. By default, the data in the external tables is deleted (truncated) after extraction. In most cases, you will accept this default.

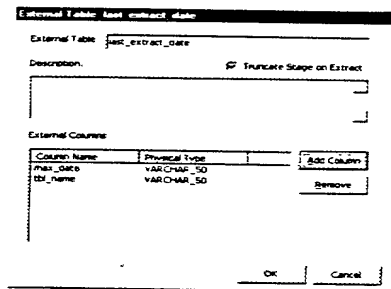


Figure 3-16 External Tables Dialog Box

To add a column:

1. Click Add Column and enter the name in the External Column dialog box.
2. Select the physical type. See Appendix E for descriptions of these physical types.

Add as many columns as necessary. After you click OK in this dialog box, the columns are added to the Column Name list in the External Tables dialog box.

Extractors

In general, SQL statements extract data and semantic instances merge data. The extractor SQL statements may be either stand-alone or extraction (pull/push) SQL statements. Stand-alone statements are typically used to produce a side-effect, such as creating or dropping indexes.

To define an extractor:

1. Choose New Extractor from the Extraction menu.
2. Enter the extractor's name and description in the Extractor dialog box (see Figure 3-17).
3. Select the appropriate input and output data stores from the drop-down lists. *EpiMart* and all data stores that are enabled for input/output appear in these lists.

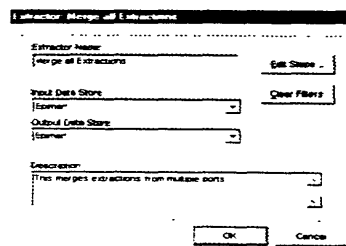


Figure 3-17 Extractor Dialog Box

4. Click the Clear Filters button to delete previous extractors from the system's memory. The effect is the same as if you were starting over, and all data will be retrieved. (See *Extracting New Rows Only*, page 41 for more information.)
5. Click the Edit Steps button to open the Extractor Steps dialog box (see Figure 3-18). The buttons on the right side of this dialog box are the means by which you define a new group, new SQL, or a new semantic instance.

The Extractor Steps Dialog Box

Figure 3-18, page 97 shows an Extractor Steps dialog box in which extractor steps have already been defined. All of the EpiCenter's extraction groups, which consist of extraction steps) appear in the right side of the dialog box.

When working with the Extractor Steps dialog box, keep these definitions in mind:

- A *job* consists of extractors, which can be shared between jobs. (A job also consists of system calls.)
- An *extractor* consists of extraction groups, which can be shared between extractors.
- An *extraction group* consists of extraction steps that are either SQL statements or semantic instances.

When extraction groups have been defined, you may select the ones for this extractor and move them to the Extractor Steps listbox on the left. Use the arrows to move them over, and the Up and Down buttons to order them in the list.

Because an extractor step must be enabled (checked) to be used in a job, de-select any extractor step that you do not want to be run. You may de-select an extractor step to restart a job that failed, or to debug a subset of the steps. In most cases, if a job fails, you should re-run the entire job.

Extraction Steps for the Group

Next, you need to define the extraction steps for the group. This group can contain any number or combination of SQL statements or semantic instances. If the step requires SQL, EpiCenter Manager provides a template for sample SQL.

To define SQL for one of the steps in this group:

1. Select its group name in the All Extraction Groups/Extraction Steps listbox, and click New SQL.

The SQL Statement dialog box is displayed (see Figure 3-19). It has three tabs, General, Description, and Store Types. (Use the description tab to add a description for your reference.)

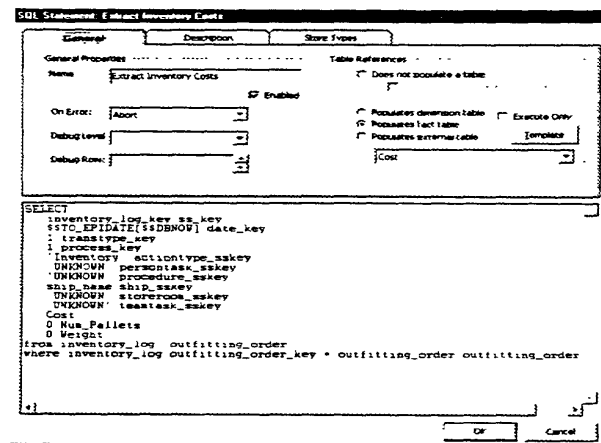


Figure 3-19 SQL Statement Dialog Box

2. In the General Properties panel of the dialog box, enter the step's name (as it will appear in the All Extraction Groups/Extraction Steps listbox). For example, a step named *Customer Raw* extracts raw customer data. The step's name must be unique.

3. You can define a step but not have it execute by de-selecting Enabled.
4. Select the action to be taken if there is an error with the step. The default is to abort the step. (You can also ignore the error, or have it print to a log.)
5. Select the debug level. These levels correspond to the verbosity levels on the **extract** command line. See *EpiChannel Debugging Levels*, page 50.
6. Select the row number at which the debug level you selected above should go into effect, if applicable. If you leave this field blank, the debug level will take effect immediately.
7. In the Table References panel of the SQL Statement dialog box, select whether the step:
 - does *not* populate a table (and if this is true, select whether it executes against input data store, or retains its default behavior of executing against the destination data store);
—or—
 - populates one of the following types of tables: a dimension table, a fact table, or an external table.

Most SQL statements are used to populate staging tables (or sometimes, external tables). A SQL statement may, however, be used to achieve a side-effect, in which case you will set it to "not populate a table." In this case, the statement is executed, and any returned results discarded.

8. If the step references a table, select the table from the drop-down list.
9. Execute Only. (*This option is not supported for this release.*)

Select this when SQL is to be expanded to reflect the structure of one of these tables, but the returned rows from the SQL statement are not meaningful. EpiChannel will not insert rows in the statement.

Extraction SQL is executed against the input data store and must be in the dialect of that database engine. The results are stored in the destination data

store. EpiChannel automatically resolves SQL dialect problems if you use the Epiphany database-vendor independent macros consistently in your SQL. (See Appendix B, *Epiphany Macros*, for more information.)

Display Sample SQL

If the step populates a table, clicking the Template button displays sample SQL for the step in the lower half of the dialog box. The SQL will need to be modified to contain the FROM and WHERE clauses appropriate for the tables in the source system, but the template lists the columns that must be returned. It does not matter in what order the columns are returned as long as they have the proper column name (and “extra” unused columns may be returned).

Restrict Data Stores for the Extraction

To restrict the input/output data stores for this extraction:

1. Specify them in the Store Types tab of the SQL Statement dialog box.
2. Click Add Type to add a type.

If a type is listed, the statement is disabled for any other type except those listed. For example, you could use the same SQL twice, once with Microsoft SQLServer syntax and once with Oracle syntax, and have one or the other statements automatically disable itself when the source database is of the wrong type. This is of value when the group that contains the SQL statement is shared by multiple extractors. Another approach to handling SQL dialect problems is to use Epiphany SQL replacement macros instead of database vendor-specific constructs. See Appendix B, *Epiphany Macros*, for more information.

Additional Steps

To create additional steps within the same group, click OK to return to the Extractor Steps dialog box, and repeat the appropriate steps as described above.

Semantic Instance

To set up a semantic instance:

1. Either create a new group or select an existing group in the Extractor Steps dialog box.
2. Click the New Semantic button. The Semantic Instance dialog box is displayed.
3. Select the fact or dimension table that the semantic instance references.
4. Select the Object from the drop-down list. An object is either a fact table or a base dimension table (the one to be operated on by the semantic type).

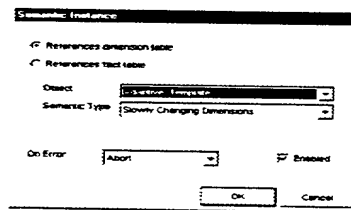


Figure 3-20 Semantic Instance Dialog Box

5. Select the semantic type from the drop-down list. See Appendix G for descriptions of semantic types.
6. Select the action to be taken upon error: abort, ignore, or print. Enabled must be selected for this action to take place.

- 7. To create additional semantic instances for this group, repeat Steps 2 through 4.

Jobs

Use the Job dialog box to order the job steps (extractors and system calls) that comprise a job. You may use the Add Job Steps dialog box (available through the Job dialog box) to create system calls.

An EpiCenter has a default job consisting of three default job steps: the Aggbuilder system call for aggregation, the End of Extraction extractor, and the Refresh system call for refreshing the Application Server. (For the Refresh system call to work, the Application Server must be configured as described in Appendix A.) You may customize the default job and create as many other jobs as necessary.

To open the default Job dialog box, double-click it. To open a new job dialog box, right-click the Jobs folder and select New Job.

The Job dialog box has two tabs: General and Job Steps (Figure 3-21).

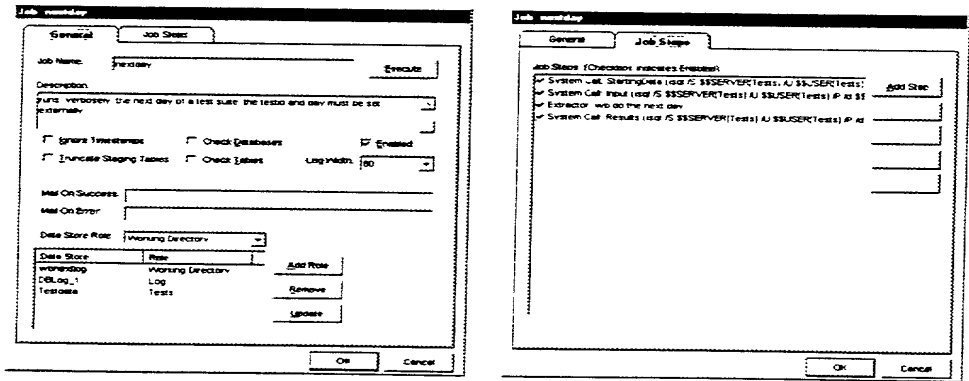


Figure 3-21 Job Dialog Box

The General tab provides options you can set that control the following aspects of a job:

- For a job other than the default one, assign a name and description.
- Enable a job. EpiChannel executes enabled jobs only.

New jobs are initially disabled because their setup is not complete. You may, however, disable a completely functional job in some circumstances to accommodate system changes. For example, if a database is in the process of being moved or repaired, jobs that populate that database could be disabled as a protection against accidental execution.

- Set up the data stores for the job (by default *EpiMart* and *JobLog*) and indicate their roles; for example, *working directory* or *log*. See *EpiChannel Output*, page 52.
- Select the width of the log, either 80 or 132 columns. Select 80 for VGA monitors and 132 for SVGA monitors.
- Assign addresses for e-mail notification of the job's success or failure. See *Configuring E-Mail*, page 106.
- Request that before executing the job, EpiChannel check that all databases referenced by any job databases and all EpiMart tables are available. Unless you are sure that this is true, select this option.
- Request that all timestamps be ignored during job execution (Ignore Timestamp).
For example, by ignoring timestamps, you could retrieve all rows, without any date filters (based on EpiChannel's special filtering used for incremental extractions).
- Turn off the truncation of staging tables. (Use this only in development mode to restart a job.)
- Execute a single job.

Click the Execute button in the Job dialog box (Figure 3-21) to execute this job. The Execute Job dialog box shown in Figure 3-22 is displayed. Use this dialog box to:

- Run the job as a trial run.
- Enter a new instance name (by clicking the New button).
- Select a debug level that corresponds to the EpiChannel (**extract**) verbosity levels (although you cannot specify row numbers). This debug level takes effect before the first SQL statement.
- Indicate the total number of rows to be transferred in a pull/push operation: all rows or the first N rows. This value is reset with each extraction statement.

Using the Execute button is equivalent to invoking the **extract** command with this job as the job option. The Execute button, however, also supplies the database name, password, and so forth, so that **extract** does not depend upon the Registry entries.

Note: If you have multiple EpiCenters, be aware that a job you execute via the Execute button in *EpiCenter A* could be run against *EpiCenter B* if you are using EpiCenter Manager to browse EpiCenter B at the time of the run. This is because the extraction run uses the default values from the Registry, and these values apply to the “current” EpiCenter. This error

will not occur if you use the **extract** command when browsing another EpiCenter.

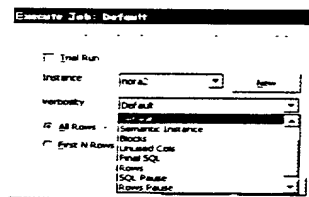


Figure 3-22 Execute Job Dialog Box: Selecting Debug Level

Use the Job Steps tab to do the following:

- Enable job steps by selecting them in the check box to the left of their names. Only enabled job steps are run.
- Change the order of job steps, using the Up, Down, and Remove buttons.
- Update the job definitions in metadata when you modify the job steps. (Click Open to open another dialog box in which you can update a job step.)

Adding Extractors and System Calls as Job Steps

You will also use the Job Steps dialog box to add one of your defined extractors as a job step and to create system calls. Click the Add Step button, which displays the dialog box shown in Figure 3-23.

To add an extractor:

1. Select Extractor.
2. Select the extractor from the drop-down list, and click OK. The extractor now appears in the Job Steps list.

As mentioned, sites with more complex databases may require multi-stages and additional commands, such as lookup tables, aggregation splits, gathering data into ranges (binning), and duplicate detection. For this purpose, you may use system calls, which are executed during a job as if invoked from the console's DOS command line.

To create a system call:

1. Select System Call.
2. Enter the name of the system call, the action to be taken upon error (abort, ignore, or print), the command line, and an optional description.
3. Click OK to add the system call as a step.

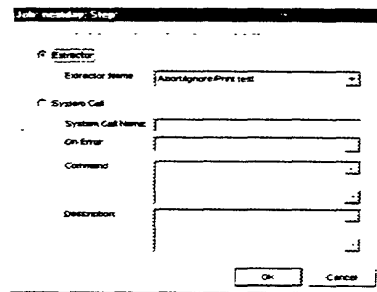


Figure 3-23 Add Job Step Dialog Box

Configuring E-Mail

To have the system automatically send e-mail notification of successful job completion or job failure, you need to set up your e-mail password:

1. Choose Configuration for the EpiCenter menu.
2. In the Configuration tab of the Configuration dialog box, select Mail Password and enter your e-mail password in the value textbox.
3. Click Update.

The Epiphany system uses the machine's default e-mail service. You may enter another mail service, if appropriate.

Truncating Tables

By default, all staging tables and external tables are truncated prior to an extraction. You can set truncation on or off via individual fact, dimension, and external table dialog boxes, or use one dialog box to define all tables in a constellation. They both refer to the same fields in the database.

To define the set of tables for truncation:

1. Choose Edit Truncation from the Extraction menu, or click the Edit Truncation toolbar icon in the toolbar menu.
2. In the Truncate Tables before Extraction dialog box (Figure 3-24), select tables to be truncated.

All of your tables are listed in columns by type (Dimensions, Facts, and External tables). Check each one in the list that you want truncated. (Clicking the All button selects all of the tables in a column; clicking None de-selects all of them.)

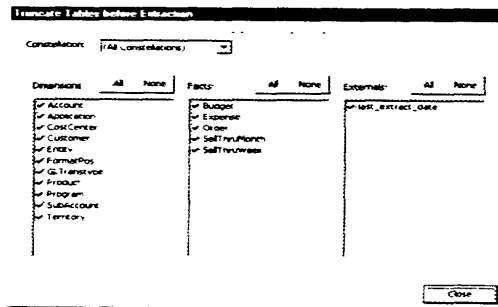


Figure 3-24 Truncate Tables before Extraction Dialog Box

Purging EpiMart Tables

You can remove tables from the EpiMart that are no longer needed. For example, you may have unused tables to delete or, as a result of metadata changes, have decreased the number of aggregates. These higher numbered aggregate tables remain in effect until you purge the EpiMart tables.

You should first try a trial run.

To purge database tables:

1. Choose Purge EpiMart Tables from the EpiCenter menu.
2. Select Trial Run in the Purge tab, and click Go.

After the run, click the Results tab to see which tables will be deleted. If these results are acceptable, return to the Purge tab, de-select Trial Run, and click Go to delete these tables from your EpiMart.

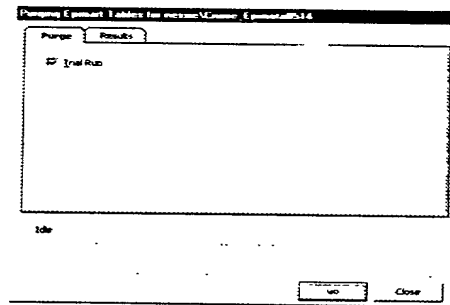


Figure 3-25 Purge EpiMart Tables Dialog Box

Security

The Epiphany system has two separate areas of security:

- *Authentication*, or the ability to log on to the system, which is determined by the Windows NT operating system.
- *Access Rights*, or what the user is allowed to access after having logged on. The Epiphany system can also use Windows NT Groups to administer access rights.

You can set access rights (or permissions) to ticksheets and Saved Queries (pre-built reports). Epiphany's metadata supports these permissions:

- The ability to use a ticksheet.
- The ability to save queries for those ticksheets.
- The ability to access data based on the values of dimensional attributes; that is, to filter database values (for example, to allow some users to see only data for the Western Sales Region).

Before Epiphany users can open any ticksheet, they must be granted access to it. Users have the ability to see and use all ticksheets to which they have access, and to all ticksheets granted access to their groups.

Note: As an administrator, you must use the security dialog boxes in EpiCenter Manager to grant access to a newly created ticksheet before any user can access it.

Users and groups also have access to pre-built reports (Saved Queries) that use the ticksheet. A saved query is simply a list of settings for the various controls that make up the Web page that is a ticksheet. (Epiphany permits global access to certain objects, such as Saved Queries, which are shared objects.) Saved queries can be granted to individual users, to individual groups (and therefore all users in the group), or to all system users (global queries).

There is the concept of a default report for each ticksheet at the user, group, and global levels. A default report is the set of ticksheet settings that a user sees when he or she first opens that ticksheet. In general, the most specific default query is used. Thus if a user-level default query exists, that query is opened in the user's Web browser. If no user query exists but a group default query exists for the ticksheet, the group default query is displayed. If neither of these exists, the global default query is displayed.

You can assign both users and groups explicit permissions for global Saved Queries (see Table 3-2).

Table 3-2 Global Query Permissions

Type	Definition
None	Cannot save global queries.
Save/No Default	Can save global queries, but cannot change which one is the default.
Save and Default	Can save global queries and can also specify which is the default (this is an administrator's role).
Inherit	(For users only) The user's ability to save global queries is determined by the groups to which he or she is a member.

If a user is explicitly given one of the settings None, Save/No Default, or Save and Default, then this determines his or her ability to save global queries. The default setting for users, however, is Inherit, which means that a user is given the access rights that are the least restrictive of the groups to which the user belongs. For example, if the user is a member of a group that has Save and Default access (the least restrictive), then this user also has that access.

You can use EpiCenter Manager to browse the Saved Queries for a ticksheet, as well as view who has access to those queries. See *Measures and Ticksheets*, page 86.

Important: Whenever a ticksheet is created via Web Builder, the administrator should save one global query and make it the default. This allows check boxes to be set to reasonable values for all users when they first use the ticksheet. Because the default global default query serves a special purpose, it does not show up in Epiphany's front-end Report Gallery, which normally shows all Saved Queries to which a user has access.

Dimension Column Access

When a user clicks the Create Report button in a ticksheet to generate a report, some results may be filtered automatically so that the report's output is restricted. Both the User and Group Definition dialog boxes have an Access Rights button, which displays the Access Rights dialog box (see Figure 3-26). Use this dialog box to restrict dimension column access. Select which dimension column to restrict, and which values to *allow* the user or group to see. The values entered in this list should correspond to actual database values in the base dimension table to which that dimension column corresponds.

For instance, selecting *Date.fy_name* as the field with the values 1997 and 1998 causes all reports to be filtered with these values.

If a user is a member of one or more groups, then all group-level restrictions apply to the user. The permissions are additive, so if Group1 has access to Fiscal years 1997, 1998, and Group 2 has access to 1996 and 1997, and a user who belongs to both groups is also explicitly granted access to 1995, then when this user runs a report, values for the years 1995 through 1998 are displayed.

A ticksheet is tied to a particular constellation, and all queries for that ticksheet live within that constellation. When a user runs a report, only those dimension column restrictions that apply to that constellation are used. For instance, if an EpiCenter has two constellations, one for Sales and one for Expense, then it does not make sense to filter on Expense dimensions when the user is working with the Sales ticksheet.

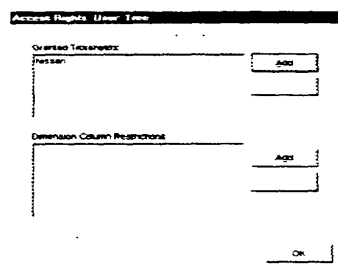


Figure 3-26 Access Rights Dialog Box

Defining Groups

In general, you should attach permissions to groups instead of users to simplify the maintenance of access rights. Users inherit all permissions from all groups to which they belong, except that a user can be denied permission to save global queries (otherwise available to members of his or her group).

You can assign a user's group membership using either the User Definition or Group Definition dialog box. Group-level queries are visible only to members of the group.

Global Queries

Global Queries are saved reports (created via ticksheets) that every user of the system can see and use. Normally, control is set by groups.

The options are—

- **Inherit.** Select Inherit if you want the user to inherit the same rights to reports as his or her group. This is the default value, and is available only for users, not for groups.
- **None.** This user cannot save group queries.

- **Save/No Default.** The user can save group queries, but cannot change which query is the default.
- **Save and Default.** The user can save group queries, as well as to specify which is the default (this is one of the tasks of a group administrator).

Note: The user's ability to save group-level queries is independent from his or her ability to save Global queries.

Synchronized Groups

For authentication purposes, new users may be added to the Epiphany metadata tables in an "autopilot" fashion via synchronized groups. You need to precede Windows NT synchronized group names with their Windows NT domain name prefix (and matching name).

When the synchronize option is set, each time a user logs in, the Windows NT security API is accessed for a list of group names to which that user belongs. Any name matching an Epiphany group name causes the appropriate group membership to be affected. If necessary (for example, the first time a user logs in), the user record itself is created. This last feature allows Windows NT administrators to create new Epiphany users.

To define groups:

1. Select New Group from the Security menu. The Group Definition dialog box is displayed (see Figure 3-27).
2. Enter the group name.
3. In the Save Global Type drop-down list, select an option as described in *Global Queries*, page 112.

4. Click Access Rights to grant group members permission for all or a subset of the available ticksheets. Select a ticksheet from the Granted ticksheet list, and click Add to give the group access to that ticksheet.

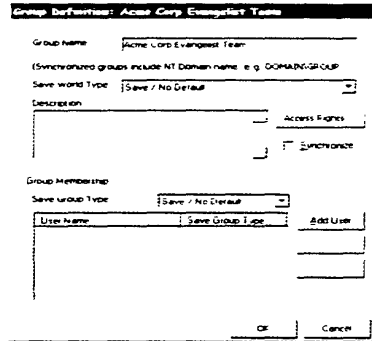


Figure 3-27 Group Definition Dialog Box

5. To restrict access rights for the ticksheet to certain rows (row-level security), click the Add button in the Dimension Column Restriction dialog box.

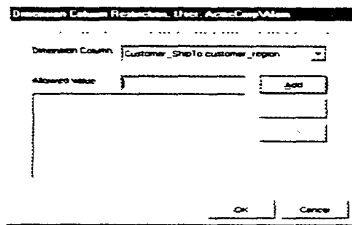


Figure 3-28 Dimension Column Restriction Dialog Box

6. Select the dimension column from the drop-down list. The dimension columns fields display in the listbox.

7. Selecting a name from the listbox displays it in the Allowed Values text box. The Add button specifies the values that a user or group is allowed to see. Select additional fields as appropriate, and click Add to add them to the list.
8. Click OK to return to the Group Definition dialog box.
9. In the Group Membership panel, select the Save Group Type from the drop-down list. The options are—
 - None. Group members cannot save their ticksheet configurations.
 - Save/No Default. Group members can save ticksheets they create, but not default queries.
 - Save and Default. Group members can save ticksheets and default queries.
10. Add users to the group. Click Add User and select group members from a list of defined users.
11. Select the Synchronize option to make group membership for the user a subset of the Windows NT memberships for that user. Synchronization is performed only on groups for which Synchronize is selected in this dialog box.

Defining Users

When new users are entered into the system, they have these default permissions:

- no access to any ticksheets.
- no group membership.
- cannot save any queries.
- can view all data in all dimensions—there are no dimensional restrictions.

To add or modify a user definition:

1. Right-click the Security folder and select New User. The User dialog box (Figure 3-29) is displayed.
2. Enter the person's username and first and last names.
The first and last names are used for display purposes only. If they are missing, then the username is displayed.
For Windows NT authentication, enter users with their Windows NT domain prefix to distinguish among users with the same name but who are in different domains.
3. In the Save Global Type drop-down list, select an option as described in *Global Queries*, page 112.
4. If tick sheets have been created, in special cases you may click Access Rights to grant that user permission to view all or a subset of the available tick sheets and tick sheet attributes. Usually, this kind of access is granted for Groups.

Figure 3-29 User Dialog Box

5. In the Access Rights dialog box, select a tick sheet from the Granted tick sheet list and click Add to give the user access to that tick sheet.

6. To restrict access rights for the ticksheet to certain rows (row-level security), click the Add button in the Dimension Column Restriction dialog box.

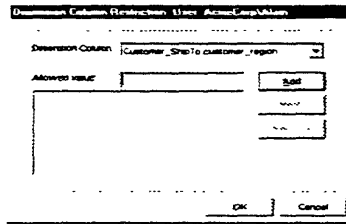


Figure 3-30 Dimension Column Restriction Dialog Box

7. Select the dimension column from the drop-down list.
8. In the Allowed Value text box, enter the name of the field whose values this user will be able to access, and click Add. Enter additional fields as appropriate, and click Add to add them to the list.
9. Click OK to return to the User dialog box.

Basic Concept: Adaptive Architecture

In traditional client/server application environments, changes to an underlying table's schema adversely affect programs that operate on that schema. The goal of Epiphany's Adaptive Architecture is to allow on-the-fly changes to the EpiMart schema while preserving the form and proper operation of the entire Epiphany Application Suite.

A schema change in EpiMart can affect these components:

- Extraction statements that populate these staging tables.
- Semantic instance programs that merge staging data with EpiMart data.
- Aggregates defined on these tables.

- Measures that refer to these tables.
- Ticksheets that refer to columns in these tables.
- Security restrictions on columns in these tables.

The use of a single metadata repository (EpiMeta) ensures that all components of the system receive notice of a change simultaneously; for example, Aggbuilder will not try to build aggregates on tables or columns that no longer exist.

During extraction, semantic Instances are SQL programs that accomplish business transformations on extracted data. Since these programs must be changed in response to schema changes, these programs are compiled on-the-fly at execution. Thus the program uses the latest schema metadata in its construction, which results in a well-formed set of SQL statements.

EpiCenter's Generate Schema command creates and modifies the EpiMart tables. The first time this operation is executed, all tables are built using CREATE TABLE statements. These tables include base dimension, fact, and external tables. Certain metadata fields are used in the actual construction of these tables.

EpiCenter Manager ensures that these fields follow the proper naming conventions for table and column names. These naming conventions are as follows:

- Base dimensions
The name of the table is taken from the name of the base dimension. Each dimension column becomes a physical column in the EpiMart table with the same name.

- Fact tables

The name of the table is taken from the name of the fact table. Each metadata fact column becomes a numeric column in the EpiMart table with the same name. Each dimension role and degenerate dimension of the constellation to which the fact table belongs become a foreign key column in the fact table.

- External tables

The name of the table is taken from the name of the external table. Each external column becomes a column in the EpiMart table with the same name.

Each time Generate Schema is executed, EpiCenter Manager records the schema of the newly built tables in a set of metadata tables called the Actual tables. When subsequent changes are made to the schema definition, EpiCenter Manager examines these Actual tables to compute a delta operation.

Important: If Generate Schema is not run and the schema definition has been altered, neither Aggbuilder nor the Epiphany Application Server will start. Both will recognize a difference between the metadata schema definition and the contents of the Actual tables.

Table 3-3 describes the schema operations.

Table 3-3 Schema Operations

Operation	Action
Add a new base dimension	The table is created from scratch.
Delete a base dimension table	The table is dropped and all dimension roles that use that base dimension are deleted.
Rename a base dimension table	Treated the same as a deletion of the old table and an addition of the new table.

Table 3-3 Schema Operations

Add a new dimension column to an existing base dimension	The existing table is altered to include that column. All existing rows take the default value for that column.
Delete a dimension column from a base dimension	The column is removed.
Rename a base dimension column	Treated the same as a deletion of the old column and an addition of the new column.
Add a new fact table	The table is created from scratch.
Delete a fact table	The fact table is dropped.
Rename a fact table	Treated the same as the deletion of the old name and an addition of the new table.
Add a new fact column	The table is modified to have the new column. All existing rows get the value 0.
Delete a fact column	The column is removed.
Rename a fact column	Treated the same as a deletion of the old column and an addition of a new one.
Add a dimension role	The column is added to all fact tables in that constellation. The value for all existing rows is set to 1, which is a special value that always points to the UNKNOWN row in the base dimension to which that role refers.
Delete a dimension role	The column is removed from all fact tables in that constellation.
Rename a dimension role	Treated the same as the deletion of the old column and an addition of the new one.

Table 3-3 Schema Operations

Add a new degenerate dimension	The column is added to all fact tables in that constellation. All existing rows get the special value UNKNOWN.
Delete a degenerate dimension	The column is removed from all fact tables in that constellation.
Rename a degenerate dimension	Treated the same as a deletion of the old column and an addition of the new one.
Add a new external table	The table is created from scratch.
Any change to an external table including changes to column	The table is dropped and recreated.

Generating Schema

After using EpiCenter Manager to define metadata, you need to convert these definitions into Actual tables (EpiMart).

To generate schema:

1. Choose Generate Schema from the EpiCenter menu.
2. Select Trial Run to see what the results will be without making any changes.
3. By default, only tables that have changed since the last time you generated schema are rebuilt.

Select Force Rebuild of all Tables only if you want to rebuild *all* of your tables.

4. Click Go (and watch the progress bar).

5. After the trial run is finished, click the Results tab to view which tables were updated.
6. If the results are acceptable, de-select Trial Run in the Schema tab, and click Go.

After the schema has been generated, the system records the current state of the metadata so that the next time the schema is updated, the current definitions can be compared with the new ones, and the appropriate tables can be created or altered as necessary.

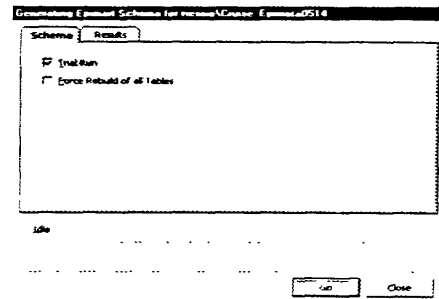


Figure 3-31 Generate Schema Dialog Box

Exporting/Importing Metadata

Epiphany provides a metadata export/import utility for moving metadata between EpiCenters and for backing up the definition of an EpiCenter. To use this tool properly, you need to understand the metadata concepts (see *Metadata Overview*, page 279).

The export operation produces a Microsoft Access database that contains a representation of the metadata that was chosen for export. Upon import, this representation is converted back into valid Epiphany metadata in the target EpiMeta. Using EpiCenter Manager, you have granular control over which

metadata objects get exported during each operation. For example, you can use the same constellation definitions, extractors, and ticksheets in several EpiCenters. Also, during import, you control whether or not to overwrite existing metadata that conflicts with what is in the import file.

Reasons for exporting files other than publishing them for import are to save files for document control (source safe) and to send files as an e-mail attachment; for example, you might e-mail an exported file to Technical Support for analysis.

You can produce a new EpiCenter by importing all or a subset of the following metadata files from an existing EpiCenter: Base Dimensions, Constellations, Measures and Ticksheets created via Web Builder, Data Stores, External tables, Extractors, SQL Templates, Jobs, Groups, and Users.

To import metadata files:

1. Right-click the EpiCenter icon and select either Import Metadata or Import Templates from the New EpiCenter menu.

An option in the import dialog box allows new data to replace existing entries of the same name. Select Save to overwrite an existing entry of the same name.

2. Select the file name from the *Choose the Name of an Import file of type* dialog box and click Open. Save the file.

See Appendix H, *Export/Import of Metadata*, for more information.

09625518.072500

Web Builder and Ticksheets

This chapter introduces the Clarity and Relevance ticksheets and explains how to use Web Builder to create these ticksheets. A ticksheet provides users with point-and-click access via a Web browser to integrated information from sources such as sales, finance, and customer support databases. (The term *ticksheet* refers to the fact that users tick, or check, items on a Web page.) Those who use Clarity or Relevance to access their data warehouse do not need to know anything about the underlying data's structure. All they need to know is what kind of information they want to analyze.

Generating a Report

Clarity and Relevance ticksheets are easy to use. (For examples of ticksheets, see Figure 4-1, page 127 and Figure 4-2, page 128.) In general, you will follow these steps:

1. Select the attributes and measurement criteria for a report.
2. Select the display options.
3. Apply filters to refine your search.
4. Click the Create Report button.

A report, or results page, displays in the browser window. If you have questions while using Clarity or Relevance, click the Help button in the left frame of the window for online help.

A description of this procedure follows.

Attributes and Measures

In Clarity, the attributes are arranged by columns and rows. For example, if you select products for the column and years for the row, the report will have columns for each of the designated products in the database and rows for each of the designated years. In Relevance ticksheets, you may select attributes in order to forecast trends, or to analyze highs and lows, best and worst cases, or graphs for various aspects of your business.

A ticksheet has one or more *selection columns* that consist of *selection column elements* (see Figure 4-1, page 127). You may select one element from each column. This combination comprises a specific *measure*, or business calculation. For example, assume you selected Units, Booked, and Net Price. The system recognizes this combination of elements as the measure *UnitsBookedNetPrice* and applies the calculations defined for this measure to aggregate the number of units booked times the net price for each product. The results are placed in each cell, arranged by product and by year. The numbers are totaled (summed) for columns and rows.

To find out what a selection column element means, click its link (see Figure 4-1, page 127), which brings up the Dictionary page where the selection column elements for an EpiCenter are defined. (Each site creates and maintains its own dictionary entries.)

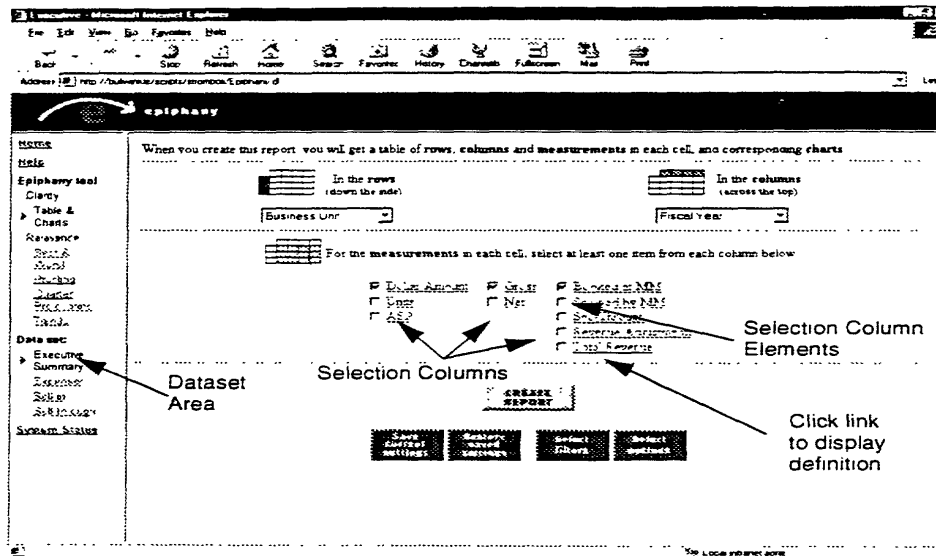


Figure 4-1 Sample Clarity Web Interface (Ticksheet)

Display Options

You can set ticksheet options that enable you to:

- select how numbers are represented.
- receive graphical charts with your report.
- limit the rows that display.
- sort rows either by value or row name.
- display data in either HTML or spreadsheet format.

Click the Select Options button on the ticksheet to open the Options window.

Filters

Rather than receiving a report with the results for all of the criteria you selected, you may limit the data by using filters. For example, you may restrict the search to the last three years, or to products sold only through direct channels.

Click the Select Filters button on the ticksheet to open the Filters window.

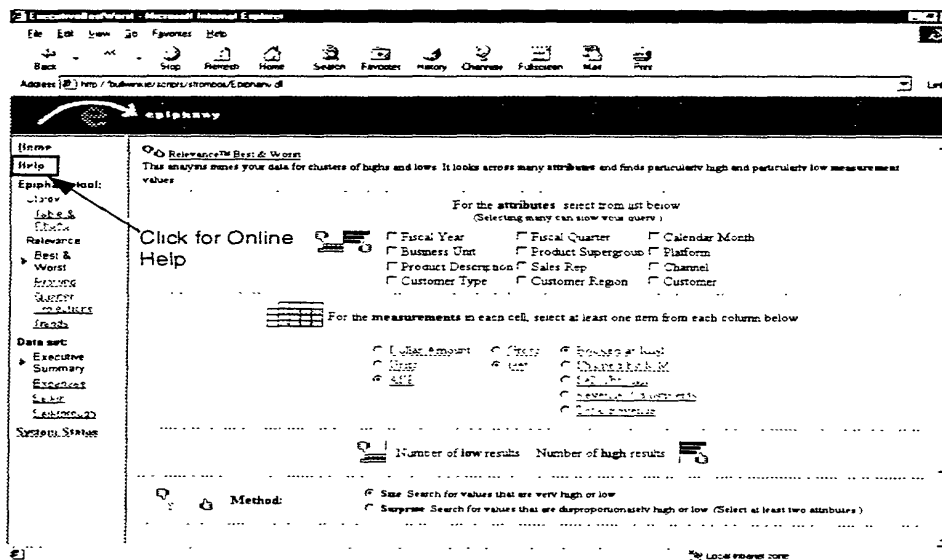


Figure 4-2 Sample Relevance Ticksheet

Web Builder Overview

The front-end ticksheets for Clarity and Relevance are created via Web Builder. Figure 4-3 shows Web Builder's system hierarchy. Each Web Builder *ticksheet* is specific to a constellation within an existing EpiCenter. Ticksheets are organized by *datasets*, which are a logical user-interface grouping of ticksheets that display similar views of data. The ticksheet's major components are attributes, selection columns, and filters.

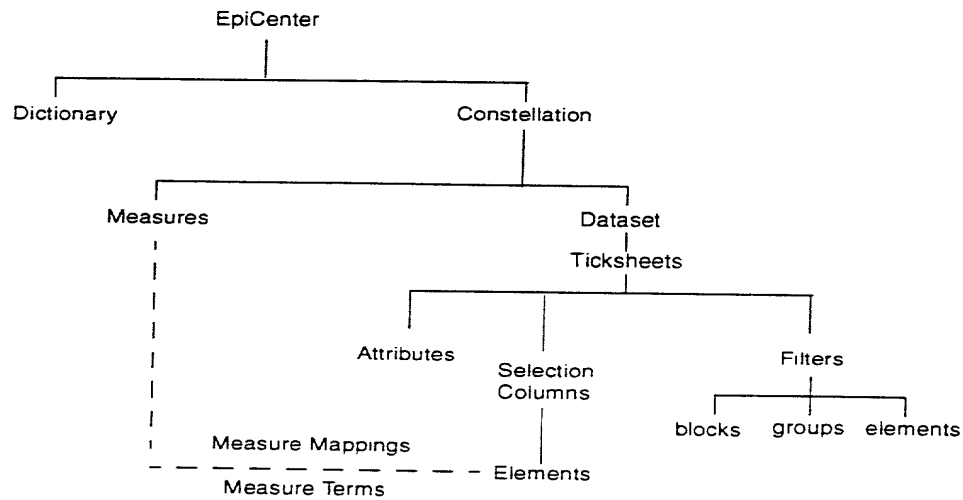


Figure 4-3 Web Builder System Overview

A *measure* is a single business calculation that is usually the result of an arithmetic combination of measure terms. A *measure term* is one component of an arithmetic expression that makes up a measure. It refers to the aggregation of a single fact column, such as *SUM(Order.net_price)* with a particular transaction type. It can be combined with other measure terms to create a composite measure. These calculations are the values returned in the cells of Clarity and Relevance reports.

Measure mapping is the association of a set of selection column elements on a ticksheet map with a specific measure. When the ticksheet user selects elements in selection columns, the system maps this combination to a measure. It then applies this measure's calculations and returns the results of the query.

Measures, as are ticksheets, are constellation wide. The dictionary entries that define measure terms to users, however, apply to the entire EpiCenter.

You will use Web Builder to define measures, dictionary entries, and ticksheet components according to the instructions that follow.

Getting Started

After launching Web Builder, the Web Builder Logon dialog box is displayed. To log on:

- Enter your server system administrator password.
- Enter the name of your server, your username, and the EpiCenter to which this ticksheet belongs.

The values you entered in this dialog box, except for your password, are saved so you do not have to enter them again the next time you log on.

The Web Builder Window

When you log on to Web Builder, the Web Builder window contains the main menu. Descriptions of the main menu, pop-up menus, and Web Builder icons follow.

The Main Menu

You will use the main menu commands to create or modify ticksheets. The main menu always displays the File, Edit, System, Ticksheet, Window, Help, and About commands. The other menus are contextual. The Attribute, Element, Filter Block, and Filter Group menus display when you are working directly with them.

Use the *File* menu to—

- reload the configuration information from the database.
- save a ticksheet.
- quit the Web Builder program.
- select recently opened ticksheets.

Use the *Edit* menu to cut, copy, paste, or perform a *smart paste*. A smart paste is a convenience feature that applies to measures only. See *Defining Measures* on page 136.

Use the *Ticksheet* menu to—

- create a new ticksheet.
- open an existing ticksheet (Figure 4-4) by double-clicking the ticksheet's name in the list of ticksheets.

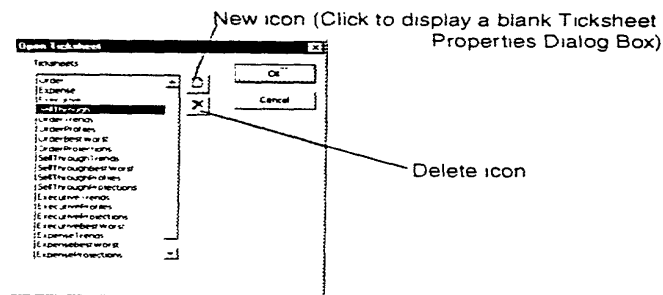


Figure 4-4 Open Ticksheet Dialog Box

Web Builder and Ticksheets

- delete a ticksheet. Select it and click the Delete icon.
- open the Ticksheet Properties dialog box for the current ticksheet. You can also click the New icon in the Open Ticksheet dialog box to display this dialog box.

Use the *System* menu to define measures, edit the dictionary, and define datasets and system properties.

The contextual menus display when you are working in the Attributes, Columns, or Filter panel of the Ticksheet dialog box:

- Use the *Attribute* menu to add and delete attributes, and to change attribute properties.
- Use the *Element* menu to add and delete elements, and to change their properties.
- Use the *Filter Block* menu to add, delete, and modify Filter Blocks. Use the *Filter Group* menu to add, delete, and modify Filter Groups. (For definitions of these terms, see *Defining Filters* on page 152.)

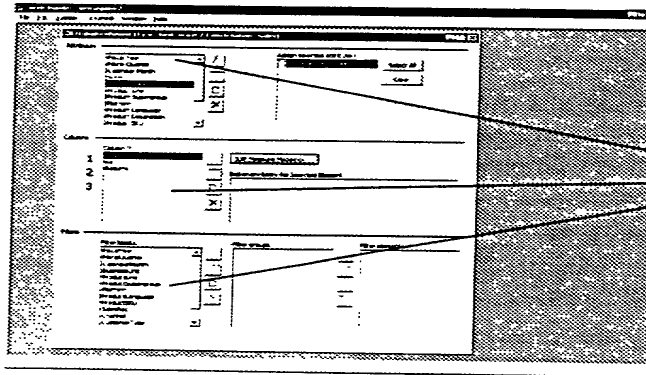
Use the *Windows* menu to select an open Web Builder window.

Open the *Help* menu to display online help for Web Builder.

The *About* menu shows the Web Builder version number and statistics for the EpiCenter as of the last time the system was loaded from the database. This includes the number of ticksheets, attributes, measures, dictionary entries, and the total number of objects.

Right-Click for Pop-up Menus

There are pop-up menus for Attribute, Element, Filter Block, and Filter Group. Right-mouse click inside the area where the attributes, elements, or filters are listed in the Ticksheet dialog box to display its pop-up menu. (The area must have items in it for these pop-up menus to display.)



Right-click
to display
pop-up
menu

Web Builder Icons

Use Web Builder icons to—

- move selected items up or down by clicking the directional arrows.
- add new items by clicking the Add a New Item icon.
- delete items by selecting the item and clicking the Delete icon.



Move
Selection
Up

Move
Selection
Down

Add a
New Item

Delete
Selection

Setting System Properties

Before creating a ticksheet, you need to set up system properties. Select Properties from the System menu, which displays the System Properties dialog box (see Figure 4-5).

Enter your company name and the GIF file name for your company logo (see Figure 4-5). This GIF file must be located in the directory named *images* in the Epiphany Web directory. Rename or copy the GIF file to **clientlogo.gif**.

Note: A logo or other mark in the upper-right corner of a ticksheet should be a GIF file that is 225 x 50 pixels. Epiphany recommends that this image be right-justified and placed on a black background. Along each edge, leave a few of the pixels black.

The dataset ordering applies after you have created more than one dataset. Use the up and down arrows to position the datasets as they should appear on the top-level page of your Web interface.

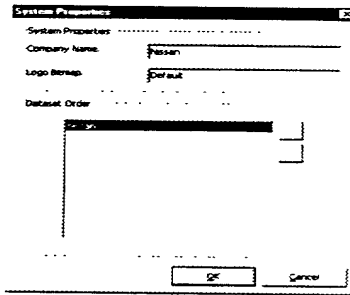


Figure 4-5 System Properties Dialog Box

Defining Datasets

A dataset is a logical grouping of similar ticksheets within a constellation. You can group related Clarity and Relevance ticksheets by dataset. In the browser window, these group names, or labels, appear in the left frame of the ticksheet under the Dataset heading. (See *Sample Clarity Web Interface (Ticksheet)* on page 127.) For example, one dataset may contain Executive ticksheets and another have Summary reports.

Use the Dataset Builder dialog box to set up dataset groupings for a constellation. (Open it by choosing Define Datasets from the System menu.) Use the Ticksheet Properties dialog box (Figure 4-9, page 144) to assign a ticksheet to a dataset.

To add a new dataset category, click the New button and enter the label name. Use the Remove and Rename buttons to modify the listing.

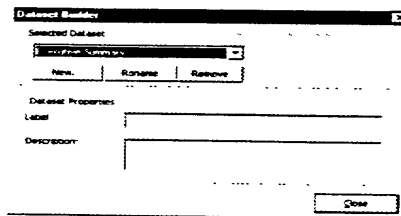


Figure 4-6 Dataset Builder Dialog Box

Defining Measures

The selection column elements that a user checks on a ticksheet will be mapped to a specific measure. You need to inform the Epiphany query machinery of the following:

- which set of selection column elements should map to a specific measure (as described in *Measure Mapping* on page 149), and
- how to convert the measure into the numeric data of the report cells. You will do this by defining measures.

As mentioned, a measure is a single business calculation that is usually the result of an arithmetic combination of measure terms. A measure term is one component of an arithmetic expression that comprises a measure, and it refers to the aggregation of a single fact column with a particular transaction type.

Use the Measure Builder dialog box to define a measure:

1. Select Define Measures from the System menu. The Measure Builder dialog box (Figure 4-7) is displayed.
2. In the Measure Selections panel, click New and enter a name for the new measure. Typically, measure names are a combination of the selection column element names; for example, the measure ASPBookedOrdersGross indicates that the user selected Average Sales Price, Booked Orders, and Gross.
3. Select the unit of measurement from the Units drop-down list. You can define these options using in the Measure Units tab of the EpiCenter Manager's Configuration dialog box (see *Measure Units* on page 247).

Example units of measurement are—

- Currency (for money units)
- Percent (for percentages)
- Units (for the count of an item)

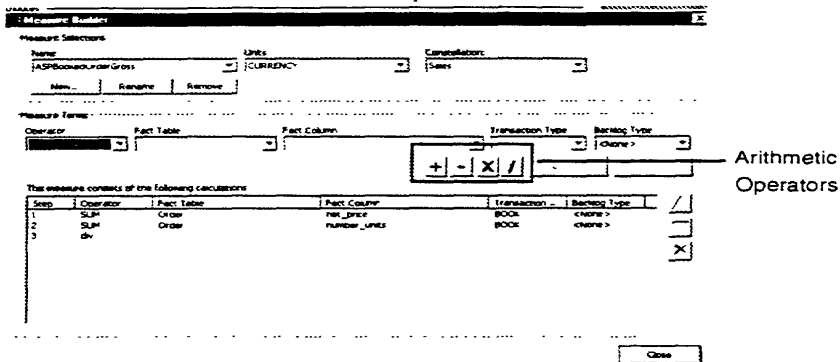


Figure 4-7 Measure Builder Dialog Box

4. Select the constellation to which the measure will belong.

Notes:

Changing the constellation for an existing measure erases all measure terms created for this measure.

Defining Measure Terms

Use the Measure Terms panel of the Measure Builder dialog box to define the steps that determine how the measure is calculated. Each step applies to a specific fact table, fact table column, transaction type, or backlog type (if applicable).

Web Builder converts these steps to appropriate SQL SELECT statements. You will use Reverse Polish Notation to construct measure definitions. See *Reverse Polish Notation* on page 140 for more information.

09625518, 072500

To define a step for a measure term:

1. Select one of the SQL operators from the drop-down list: SUM, MIN, MAX, AVG, COUNT, COUNT DISTINCT (or the negative values of these operators, such as -SUM or -MAX).
2. Select the fact table where the data resides, the fact column, transaction type, and backlog type.

The backlog types are BEGIN or END; or use <None> if the backlog type does not apply. Use a backlog type when a measure term (a single line of the measure definition) should exhibit accumulating behavior. Normally, when running a report of Sales by Month, for example, the report shows only the sum of transactions that occur in each month of the report. When a backlog type is used, however, the columns of the report show accumulated values from previous months, in addition to the current month. Use BEGIN to show the accumulated value at the beginning of each period, and END to show the ending accumulated value.

For example, assume that report of sales by month transactions shows \$10 for June, \$20 for July, and \$40 for August. A report of the beginning backlog shows the accumulated value at the beginning of each month:

June	July	August	September
\$0	\$10	\$30	\$70

A report of the ending backlog shows the accumulated value at the end of the month:

June	July	August
\$10	\$30	\$70

Notes:

You can cut, copy, and paste measures using the Edit menu. The Smart Paste command performs a standard paste, and allows you to substitute values in the Fact Table, Transaction Type, or Backlog Type fields. For example, a measure that relates to the Orders fact table can be copied and pasted to apply to a Budget fact table. You can copy and paste a measure defined for the transaction type Book, and then substitute a Ship transaction type.

See Appendix C for more information about transaction types.

3. Click Add to add this as the first step in the lower panel.
4. If appropriate, add another measure term by repeating Steps 1 through 3.
5. Click the appropriate arithmetic operator to be applied to the measure terms: add (+), subtract (-), multiply (x), or divide (/). (See Figure 4-7, page 137 for the location of these operators.) See *Reverse Polish Notation* on page 140 for examples.

To change an existing measure term, select it from the Name drop-down list and modify as described above. Click Update to save the changes, which will be reflected in the dialog box.

Use the Remove button to delete a measure, and the Rename button to rename a measure.

Reverse Polish Notation

The Measure Terms panel in the Measure Builder dialog box uses Reverse Polish Notation³ (RPN) to construct measure definitions. RPN operations are performed in a last-in, first-out (LIFO) basis. All of the values to be operated upon are placed in a stack. Then the top two are operated upon and the result of that operation is placed in the stack, replacing the previous two values. Then the next top two are operated on and the result placed in the stack, and so forth.

For example, using Reverse Polish Notation for this calculation:

$1 + (2 * 3) = 1, 2, 3, *, +$ in RPN

means that 1, 2, and 3 are placed in the stack. The last two items (3 and 2) are multiplied, and the resulting value 6 is placed in the stack. The stack now holds 6 and 1, which are added, and the result 7 is placed in the stack.

In contrast, applying RPN to the calculation:

$(1 + 2) * 3 = 1, 2, +, 3, *$ in RPN

means that 1 and 2 are placed in the stack. These items are added, and the result 3 replaces them. Next, the value 3 is placed in the stack (the stack now holds 3 and 3). These items are multiplied and the result is 9.

Example 1 below shows how RPN is used to define a measure definition (in the Measure Builder dialog box). The Average Sales Price (ASP) for Booked/Gross Orders equals the total number of dollars received, divided by the total number of units shipped. This is represented as the following SQL SELECT statement:

```
SUM (Order.net_price)
SUM (Order.number_units)
div
```

Example 1

³ Reverse Polish Notation is named for its Polish inventor, Jan Lukasiewicz.

This SELECT statement is calculated using RPN as follows. The sum of all the Order net prices is calculated and placed in the stack. Then the sum of all the number of units is calculated and placed in the stack. Next, the division operator is applied to the top two items in the stack. It takes the next item in the stack, the total net price received, and divides it by the total number of units shipped, and the result equals the ASP.

For the operators that apply to the aggregates, use the arithmetic operators: add, sub, mult, and div.

Example 2 shows a similar calculation. This time the ASP is calculated for booked net orders (booked orders minus returns). Here the sums of two Order net prices are added (the returned items are represented as a negative number) and placed in the stack. Then the sums of two Order number of units (the returned items are represented as a negative number) are added and placed in the stack. The remainder of the calculation is the same as for Example 1 above.

```
SUM (Order.net_price) BOOK
SUM (Order.net_price) BOOK_RETURN
add
SUM (Order.number_units) BOOK
SUM (Order.number_units) BOOK_RETURN
add
div
```

Example 2

Defining Dictionary Entries

Selection column elements in Clarity and Relevance ticksheets, such as Units, Gross, and Sell-Through, are hyperlinked to a dictionary page. When a user clicks a link, a dictionary page displays which defines it. These dictionary entries help users understand your terminology.

Selecting Edit Dictionary from the System menu displays the Dictionary Builder dialog box (see Figure 4-8).

To add a dictionary entry, click the New button and enter the selection column element's name in the Name dialog box, and click OK. Enter the definition in the Entry Text area of the Dictionary Builder dialog box. Click Close to save the entry.

To remove an entry from the Web Builder Dictionary page, select the entry from the Entry Name drop-down list and click the Remove button.

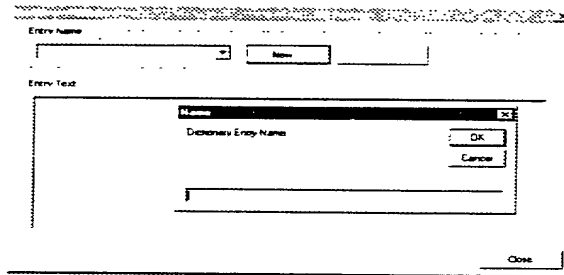


Figure 4-8 Dictionary Builder Dialog Box

You can also use the Column Elements Properties dialog box (Figure 4-12, page 149) to add and modify dictionary entries for a selection column element. To open this dialog box, double-click the selection column element, or select it and choose Element Properties from the Element menu. Click Dictionary Editor to open the Dictionary Builder dialog Box.

Note that a ticksheet selection column element's dictionary entry (if any) displays in the Columns panel of the Ticksheet dialog box when selected.

Creating a New Ticksheet

Important: Creating a ticksheet does not automatically make that ticksheet available to end users. You must first use EpiCenter Manager to grant this permission to one or more Epiphany users and groups.

Each time a ticksheet is created via Web Builder, the administrator should use EpiCenter Manager (see *Global Queries* on page 112) to save one global query and make it the default. This allows check boxes to be set to reasonable values for all users when they first use the ticksheet. Because this global default query has this special purpose, it does not show up in the Report Gallery. (The Report Gallery is a feature of Clarity and Relevance that lists the Saved Queries that a user may access.)

You may start building your ticksheet from scratch as described below. For every ticksheet, you will define attributes, columns, and filters. After you complete a ticksheet and save its configuration, it is available for a front-end user to open in a Web browser.

To create a new ticksheet, choose New from the Ticksheet menu. Use the panels in the Ticksheet Properties dialog box (Figure 4-9, page 144) to define the ticksheet.

- Ticksheet Properties

Enter the name of the ticksheet (no spaces are allowed), the label name (the name that the user sees in the Web browser), and any descriptive text.

- Display Options

Select the ticksheet type. See *Ticksheet Types* on page 145 for a description of each.

Select the number of columns that will appear on the Clarity ticksheet (Figure 4-1, page 127). A column consists of selection column elements.

- Constellation and Dataset

Select the constellation for this ticksheet from the drop-down list.

Select the dataset that the ticksheet belongs to. Datasets are subsets of the constellation's ticksheets listed in the left margin of Clarity and Relevance Web pages (see Figure 4-1). See *Defining Datasets* on page 135 for more information.

- Data Copy

You may copy attributes, columns, or filters from other ticksheets within this constellation. This is a fast way to create a ticksheet when it shares a subset of another ticksheet's properties.

Note: If you want to copy columns, be sure that the ticksheet you copy columns from has the same number of columns as your new ticksheet.

After you complete the dialog box and press OK, a new ticksheet with these properties is displayed.

The screenshot shows the 'Ticksheet Properties' dialog box. It has several sections: 'Ticksheet Properties' with fields for Name, Label, and Description; 'Display Options' with a 'Ticksheet Type' dropdown set to 'Data_Mark' and a 'Number of Columns' spinner set to 1; 'Constellation & Dataset' with dropdowns for 'Constellation that this ticksheet belongs to' (set to 'Sales') and 'Dataset that this ticksheet belongs to' (set to 'Sales_Executive'); and 'Data Copy' with checkboxes for 'Copy Attributes From', 'Copy Columns From', and 'Copy Filters From', each with a corresponding dropdown menu. Annotations on the left point to the 'Number of Columns' spinner with the text 'Select Number of Columns' and to the 'Dataset that this ticksheet belongs to' dropdown with the text 'Assign Dataset'.

Figure 4-9 Ticksheet Properties Dialog Box

09625518 072500

Ticksheet Types

The ticksheet types you may select in the Ticksheet Properties dialog box are:

- **Best & Worst**
Relevance Best & Worst searches thousands of Clarity tables automatically and builds a list of the most interesting results within those tables.
- **Clarity**
A Clarity ticksheet produces reports for selected attributes (the columns and rows of the report) with measure values inside the cells. Corresponding bar and pie charts are also provided.
- **Momentum**
[Not available in this release.]
- **Profiles**
Relevance Profiling shows a page of thumbnail charts that help you quickly understand your data.
- **Quarter Projections**
Relevance Quarter Projections answer the question, "Given where I am today, and given the history of previous quarters, how is this quarter likely to end? Will I meet my goals?"
- **Trends**
Relevance Trends fits a smooth curve to time-varying data, and forecasts one or more periods into the future.

The Ticksheet Dialog Box

The ticksheet dialog box (Figure 4-10) has three panels: attributes, columns, and filters. These sections correspond to the attribute, column, and filter areas in Clarity and Relevance ticksheets.

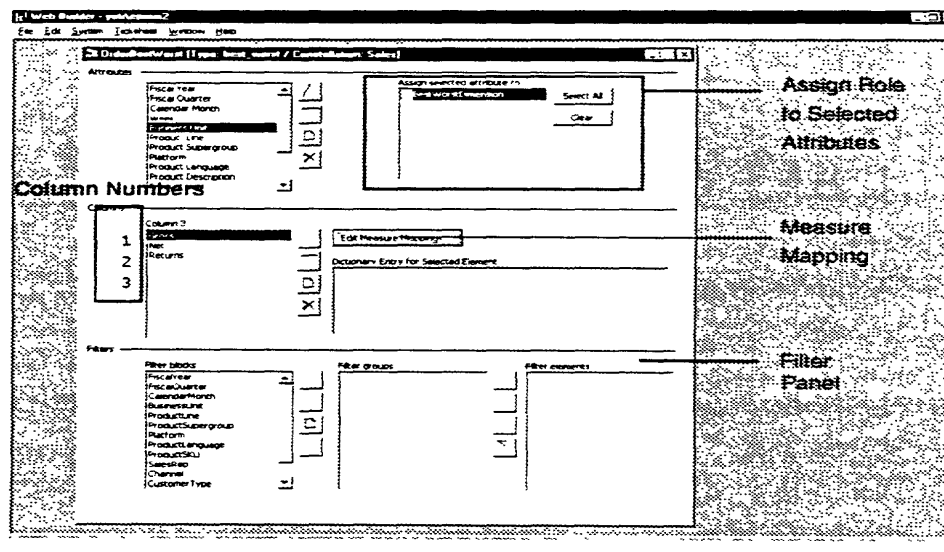


Figure 4-10 Ticksheet Dialog Box

Defining Attributes

Attributes from the dimension tables correspond to the columns and rows in a Clarity report. Attributes are also used as filtering criteria. You will use the Attribute Builder dialog box to add attributes to a ticksheet.

Follow these steps:

1. Click the New Item icon to the right of the Attributes list in the Ticksheet dialog box (or choose Attribute from the Attribute menu). The Attribute Builder dialog box shown in Figure 4-11 is displayed.
2. Enter the attribute's Label (the name that appears on the ticksheet).
3. Select the dimension column for this attribute from the base dimension table listing. (Right-click a plus sign to expand the tree.)
4. You may use the attribute as a link. For example, companies in a customer list could be linked to Web sites, such as <http://www.co-select.com>.

Use the string \$val\$ to indicate where the attribute is to be substituted; for example: [www.\\$val\\$-select.com](http://www.val-select.com).

5. Click OK to add the attribute. It now appears in the Attributes box in the panel. Follow the same procedure to create (or modify) additional attributes.
6. After defining all of the attributes, you may select an attribute and assign it an attribute role, such as row only or column only for Clarity ticksheets. (The ticksheet type determines these options.) Use the Assign Selected Attributes area of the Ticksheet dialog box (see Figure 4-10, page 146).

Important: Attributes can be defined only once per ticksheet. If you want a single attribute to appear in both the rows and columns listboxes of Clarity, you must assign it both of these roles: Clarity Row and Clarity Column.

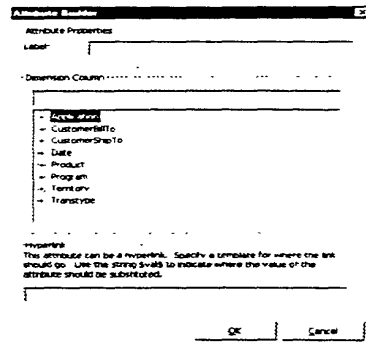


Figure 4-11 Attribute Builder Dialog Box

Defining Column Elements

The number of selection columns that you define for a ticksheet in Web Builder corresponds to the number that will display in the measurement section of a ticksheet (under the heading *For the measurements in each cell, select at least one item from each column below*).

A selection column consist of selection column elements. It is the combination of one element from each column that will be mapped to a single measure. The calculation of this single measure (applied to the data in the data warehouse) produces the values that displays in the cells of a Clarity or Relevance results page.

After using the Ticksheet Properties dialog box (see Figure 4-9, page 144) to enter the number of column elements, you can define elements for each column:

1. Click the column number (see Figure 4-10, page 146). (If elements have been defined, they will appear in the list.) Click the New icon to display the Column Elements Properties dialog box.
2. Enter the name for the selection column element. The label and abbreviation fields are automatically filled in with this same name, which you can edit.
3. Click the Dictionary Editor to add a dictionary entry. See *Defining Dictionary Entries* on page 142.
4. Click OK to add the element to the column.

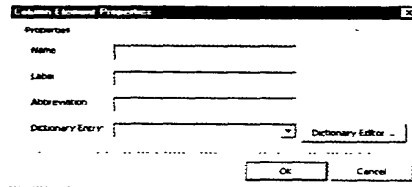


Figure 4-12 Column Element Properties Dialog Box

Measure Mapping

Measure mapping is the means by which the Epiphany system knows which measure to apply when a ticksheet user selects a set of column elements. As part of configuring a ticksheet, you need to inform the system which combination of selection column elements maps to a specific measure.

To set up measure mapping:

1. Click Edit Measure Mappings in the Ticksheet dialog box (see Figure 4-4, page 131) to open the ticksheet's Measure Mappings dialog box (Figure 4-13).

2. Select the selection column elements that comprise the measure from the drop-down list on the left.
3. Select the associated measure name from the listbox on the right. This measure is highlighted.
4. Continue to map each set of selection column elements to its associated measure.

After you close this dialog box, this measure will be invoked whenever a user selects this combination of selection column elements in a ticksheet.

After you set up the mapping, you can click the Next (and Previous) buttons to cycle through all the combinations. Clicking a measure name (in the listbox on the right) displays the set of selection column elements that comprise it.

Note: Measure mappings must be set for each ticksheet. You may use the Data Copy panel of the Ticksheet Properties dialog box to import measure mappings from other ticksheets.

Defining Column Elements

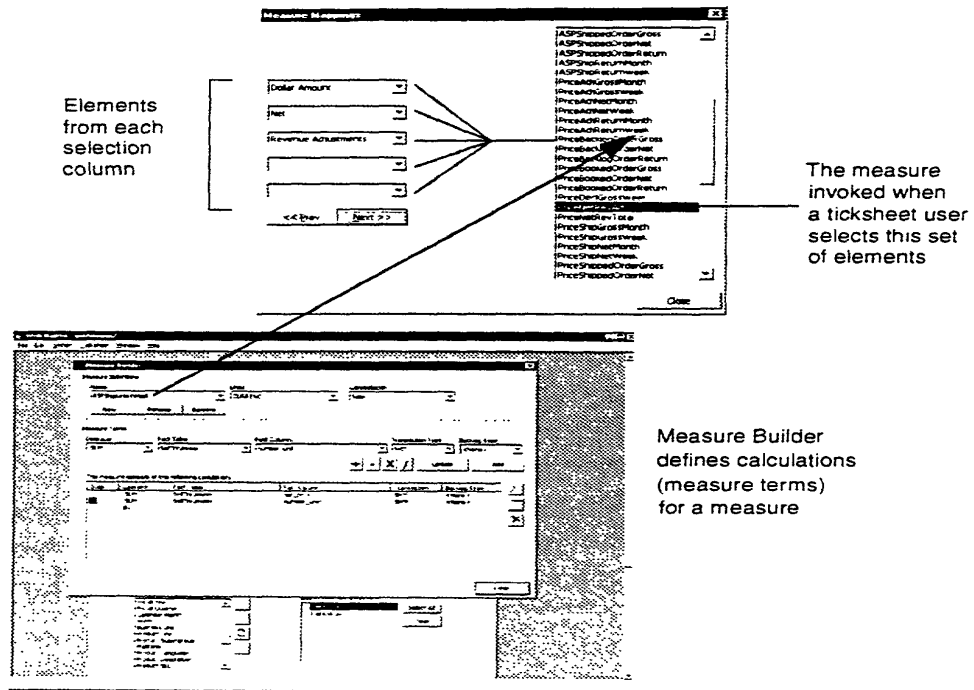


Figure 4-13 Measure Mapping

Defining Filters

Applying filters to a ticksheet query restricts the data that is accessed for that query. The Filters panel of the Ticksheet dialog box consists of Filter Blocks, Filter Groups, and Filter Elements that correspond to the same areas on a ticksheet. When you select an item in a Filter Block or Filter Group, the correct Filter menu displays in Web Builder's main menu.

There are three levels of filtering: the top, middle, and bottom level.

- Top level: A *Filter Block* is the attribute filter to be applied to the data, such as filtering the data by year, month, or customer. On a ticksheet, a Filter Block controls the user's ability to filter on a single dimension column (for a dimension role of the ticksheet's constellation). The Filter Block also defines the appearance of the filter (for example, check box versus listbox).
- Middle level: A *Filter Group* is a logical grouping of filter elements within a Filter Block. It is applicable only to check box Filter Blocks.
For example, a Filter Block by year may have the Group Q197 (first quarter of 1997), with the months January '97, February '97, and March '97.
- Bottom level: A *Filter Element* is a single check box for filtering within a Filter Group, or a single entry within a listbox Filter Block.

To edit an item in a Filter Block or Filter Group, double-click it, which displays its Properties dialog box.

After you define filters for a ticksheet in Web Builder, they become options for the front-end user to select in the Filters window of a ticksheet.

Defining Filter Blocks

A Filter Block may be a text area, a check box, a static or dynamic listbox, or radio buttons. The contents of a dynamic listbox are updated when the Application Server starts. (Dynamic listboxes are refreshed while static listboxes never change.)

Use the Filter Block Properties dialog box to create a Filter Block:

1. In the Ticksheet dialog box, click the New icon for Filter Blocks, or choose Add Filter Block from the Filter Block menu. (Double-clicking an existing item in the Filter block area allows you to modify that item.) The Filter Block Properties dialog box is displayed.

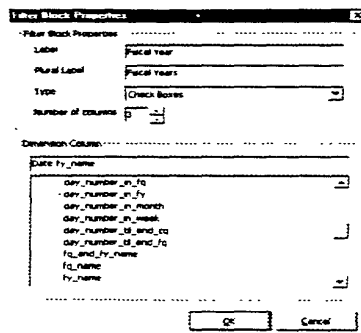


Figure 4-14 Filter Block Properties

2. In the Label textbox, enter the name for the heading on the ticksheet, such as Fiscal Year.
Note: The database values are mapped to the label on both the ticksheet and the report, or results page.
3. In the Plural Label textbox, enter the plural of the label, which is what follows *All* as in All Fiscal Years on the ticksheet.
4. Select the Type from the drop-down list: check boxes, listbox (a static listbox), dynamic listbox, radio button, or text area.
5. For the check box type, select the number of columns that the attributes will be divided into; for example, Fiscal Quarter has four columns. For other types, this choice determines the height of the listbox or text area.

6. Select the dimension column that contains the attribute.
7. Click OK to add the Filter Block.

Defining Filter Groups

Important: Grouping applies to check box Filter Blocks only. You cannot define a Filter Group for a filter unless it is a check box filter.

A sample check box Filter Grouping follows:

Group 1 — 1996: Q196, Q296, Q396, Q496

Group 2 — 1997: Q197, Q297, Q397, Q497

Group 3 — 1998: Q198, Q298, Q398, Q498

In all other cases (non-check box Filter Groups), a group is created automatically. This group's only purpose is to contain either the items in a listbox (a container for one group at a time), or the SQL for the query that makes up a dynamic listbox.

Note: Check box Filter Groups provide a convenience for ticksheet users. Creating a Filter Group does not modify the database.

To define a Filter Group, follow these steps:

1. In the Ticksheet dialog box, click the New icon for Filter Groups, or choose Add Filter Group from the Filter Group menu. The Filter Group Builder dialog box is displayed (see Figure 4-15).
2. To enter names manually, enter a new item label (for the name on the ticksheet) and its value; for example, 1996 for the label and 1996 for the value. Click Add. The new item appears in the list as 1996 [1996].

3. To edit an item, simply click it. Its contents will be displayed in the New Item Label and New item value text areas for editing. Click Update to save any changes.

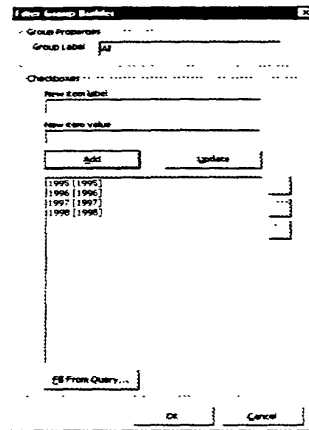


Figure 4-15 Filter Group Builder

4. To have the system fill in the list using an SQL SELECT statement (to extract a field from a table), click Fill From Query. The SQL Query dialog box (Figure 4-16) displays.

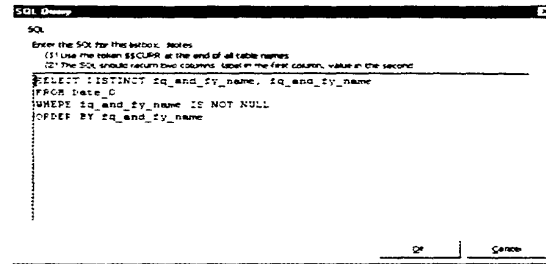


Figure 4-16 SQL Query Dialog Box

5. To have Web Builder create a template SQL for your filtering query, replace the column name and the table name for your data. Append the token `__0$$CURR` at the end of the table name; for example, `Product_0$$CURR`.
6. Click OK to begin the extraction.

The query provided by Web Builder has two columns by default: one for label and one for value.

Configuring Relevance Ticksheets

The configuration of a Relevance ticksheet is similar to that of a Clarity ticksheet. You can speed up the process by first creating a Clarity ticksheet, and then using the Data Copy panel of the Ticksheet Properties dialog box (Figure 4-5, page 134) to copy attributes, columns, and filters.

The only significant difference between Relevance and Clarity ticksheets is the attribute roles. Each Relevance ticksheet has its own attribute roles. The Best & Worst ticksheet has only one kind of role, a `BestWorstDimension`, and only attributes assigned this role appear in the ticksheet attributes area.

In general, there is no reason to conceal attributes from users. The exception is when there are so many that the ticksheet becomes cluttered, or when an attribute has extremely high cardinality (many values). For queries involving such attributes, the Best & Worst query may be very slow.

The Profiling ticksheet also has only one role, the ProfilingDimension. Again, only attributes assigned this role will appear on the ticksheet. Profiling is not effected by high-cardinality attributes as much as Best & Worst, so avoiding clutter is the main reason to not assign attributes to the ProfilingDimension role.

The Trends ticksheet has two roles, TrendsRow and TrendsColumn. The TrendsColumn role should be assigned only to time attributes, such as Fiscal Quarter, Fiscal Year, and so forth. The Trends Row role may be assigned to any non-time attributes.

The Quarter Projections ticksheet has two roles, ProjectionRow and ProjectionColumn. The attributes for ProjectionRow should be time attributes only, such as Fiscal Quarter or Month. The attributes for ProjectionColumn should be relative time attributes only, such as Days until End of Fiscal Quarter, or Weeks until End of Fiscal Quarter, or Days until End of Month.

09623518.072500

Epiphany Application Server

The Epiphany Application Server is the component of the Epiphany Application Suite that processes all user requests and returns query data in HTML format. All Epiphany applications, such as Clarity and Relevance, run on top of the Application Server. The Application Server has a multi-threaded design that allows several interactive connections to occur simultaneously.

In general, an Application Server interaction begins when the user points a browser at a Web URL address. The URL is first processed by the Web server (IIS 3.0), and then routed to the Epiphany proxy, **Epiphany.dll**. The proxy packages the request and sends it to the Application Server via a TCP/IP socket. The proxy waits for the Application Server to process the request and return a result. The proxy then takes the result and returns it to the user's browser via IIS. The proxy serves to separate the Application Server from the IIS process. This architecture allows several IIS machines to point at the same Application Server. The Application Server listens for requests, dispatches them to work threads, then processes the requests and returns the results. Figure 5-1 shows the role of the Application Server in the Epiphany system.

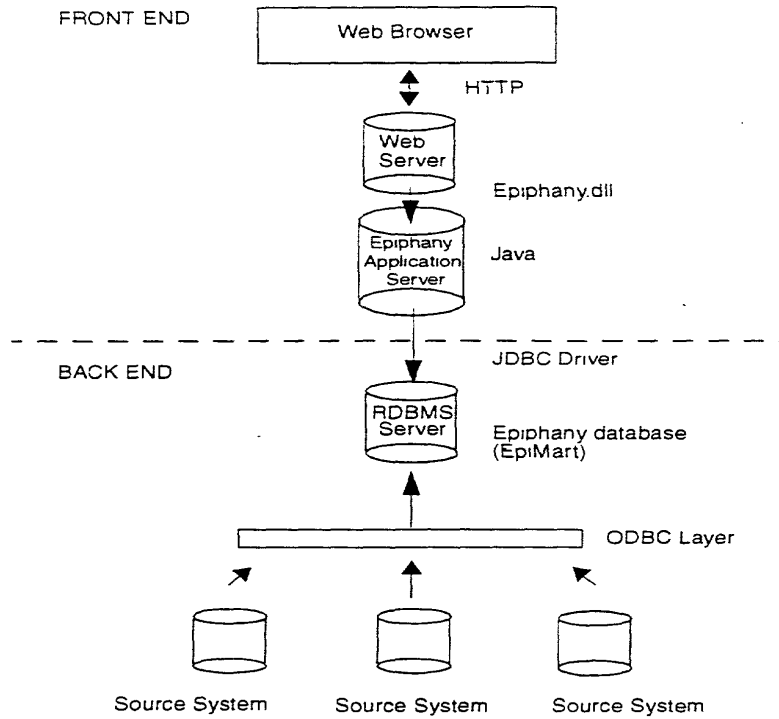


Figure 5-1 Epiphany System Diagram

The Application Server is a Java application that is normally run as an NT Service. Java is not used on the client (browser) side.

09625518.072500

This chapter covers the following topics related to the Epiphany Application Server:

- Starting and stopping (see page 162)
- Command-line arguments (see page 167)
- Refreshing (see page 167)
- EpiAppService program (see page 172)
- Epiphany proxy (see page 178)
- Logging (see page 181)
- Security (see page 186)
- Configuring output templates (see page 192)
- Customizing error messages (see page 197)

For instructions on how to troubleshoot the Application Server, see Appendix I. This appendix also describes the Application Server error messages.

Starting and Stopping the Server

This section gives instructions for starting and stopping the Application Server when you run it as a service, or as a console application.

Running as a Service

A standard Epiphany software installation (as described in *Installing the Epiphany Software* on page 213) installs the Epiphany Application Server as an NT service. You can start and stop the service using the Windows *Control Panel\Services* menu. Follow these steps:

1. From the *Start* menu, choose *Settings\Control Panel\Services*.
2. Select the Epiphany Application Server. It is usually installed under the same name as the *instance_name* you specified during installation.
3. Click Stop to stop the Application Server.
4. Click Start to start it.

Determining if the Application Server Is Running

When you run the Application Server as a service, it may be difficult to tell whether it is running. To determine if the Application Server is running:

1. Check the NT Event log by going to the *Start\Programs\Administrative Tools (common)\Event Viewer* and opening the *Log\Application* menu.

If the Service was started and is running, there will be an entry with Source = EpiAppServer. The message reads Epiphany Appserver <instancename> message: The Service was started.

2. Check for the existence of the Application Server's log file.

The Application Server creates a log file immediately upon initialization, which will be named using the following convention:

1998-06-08_14-48-15-32SRV.txt

Starting and Stopping the Server

where *1998* is the year, *06* is the month, and *08_14-48-15* is the day, hour, minute and second when the Application Server started running.

The log file resides in the logging directory, which is specified in the Windows Registry under the key:

HKEY_LOCAL_MACHINE\Software\Epiphany\Instances\instance_name\SystemLogDir

Normally, this log file is located under the directory:
instance_name\web\WWWROOT\logfiles.

If you have started the Application Server and it has been initialized, this log file resembles the one in Figure 5-2.

00625518.072500

Epiphany Application Server

```
Adding session: __ global __|0
[EpiCenter] Initializing instance test0
  EpiMeta DSN : acme
  EpiMeta Username : sa
  EpiMart Database : Acme2_EpiMart
  EpiMart Username : sa
  [TimeNav] Initializing instance test0
  [TimeNav] Initialized instance test0 in 2133 ms
[EpiCenter] Closing connections.
[EpiCenter] Initialized instance test0 in 11837 ms
AggNav initializing instance test0
AggNav initialized instance test0 in 1161 ms
Security manager initializing instance test0
Security manager initialized instance test0 in 130 ms
Save restore manager initializing instance test0
Save restore initialized instance test0 in 41 ms
[om] Beginning init in D:\TEMPL\
[om] Processed 47 templates.
Server Host: localhost
Server Port: 8081
Server Inst: test0
**** Epiphany AppServer test0 awaiting connections... ****
```

Figure 5-2 Sample Application Server Log File

Running as a Console Application

Usually, you will start Epiphany Application Server from the *Start* menu: *Settings\Control Panel\Services* menu. During debugging and the initial setup, however, it may be expedient to start the Application Server from a console window because all of the logging information is copied to the console.

You can manually start the Application Server by using the **jre** program (in your local path) from the directory:

C:\Program Files\Epiphany\instance_name\classes.

Starting and Stopping the Server

Enter the following command line:

```
jre -mx64M -classpath .;.\connect;.\connect.jar;.\
\EpiAppServer.jar;C:\PROGRA~1\
JavaSoft\JRE\1.1\lib\rt.jar com.epiph-
any.server.Server localhost:8081 @instancename
```

Notes:

If the **jre** program directory is not in your local path, you will have to prepend the pathname to the **jre** command. Normally, **jre** is installed in the *C:\Program Files\JavaSoft\jre\1.1\bin* directory. This command example also assumes that the Application Server is running on the local machine on port 8081.

For security to function properly, the user who runs this command line must have certain NT user rights. Otherwise, no one will be able to log in (you will receive *EpiLoginException* violations).

When you manually start the Application Server using the **jre** command, a console window that echoes all of the logging information is displayed on your screen. The window must remain open while the Application Server runs. If you close this console window, the Application Server terminates. If you log out of your machine, the console window closes, and the Application Server terminates.

For Authentication to Work

For authentication to work, run the Application Server as a service under the Local System account, or under an administrator account that has special NT privileges. The special NT privileges are—

- Act as part of OS
- Increase quotas
- Replace a process level token

Epiphany Application Server

The Local System account already has these privileges, and almost always, installation will be set up to run the Application Server under the Local System account.

To run the Application Server from the command line, follow these steps:

Note: This procedure affects only the local machine, not the NT network.

1. Log in as a user who has local machine administration access, in addition to the special privileges listed above.
2. Assign special NT privileges to a user account. (You must be an administrator on the machine running the Application Server.) To do so, start User Manager and enter the local machine name in the Select Domain dialog box.
3. Choose Policies/User Rights from the menu.
4. Click Show Advanced User Rights and assign the account of the currently logged-in user to have the privileges specified above.
5. Reboot your machine for the privileges to take effect.

Command-line Arguments

The Application Server's main routine is located in the **com.epiphany.server.Server** class. The Application Server takes the following command-line arguments

```
jre -classpath classpath com.epiphany.server.Server  
      machinename:port number @instancename DEBUG=1
```

If the machine name and port number are not specified, then they are read from the Registry under the *instance_name* directory. See *The Application Server's Registry Keys* on page 175 for more information. If the *instance_name* is not specified, then the server will read the following file to determine the default *instance_name* to use:

HKEY_LOCAL_MACHINE\Software\Epiphany\Instances\default key

The DEBUG=1 parameter will cause the Application Server to log extra information. This extra information includes the data about the "clean-up" routines that handle timed-out sessions and old command threads that have finished.

Refreshing the Application Server

Upon startup, the Application Server reads metadata and Registry information and caches it for use during normal query processing. The metadata that is read includes

- The Windows Registry
- The *config_master* table in the EpiMeta database
- Aggregate navigation information
- Time navigation information
- The templates used to render Clarity and Relevance result (report/chart) pages

Epiphany Application Server

- Security Information
- Ticksheet information

When any of the following events occur, you need to restart or refresh the Application Server:

- After an extraction, either the *current_datamart* has changed or there has been a change in a table that is used in a dynamic listbox filter.
- EpiCenter Manager was used to alter the EpiMart.
- Aggregates have been built or rebuilt since the last time the Application Server was started.
- Web Builder was used to add or modify a ticksheet, measure, or dataset.
- The Windows Registry entries under the following entry have changed since the last time the Application Server was started:
HKEY_LOCAL_MACHINE/Software/Epiphany/instance_name
- EpiCenter Manager was used to change security permissions for a user or group.

Note: The fact that a template file has changed does *not* require a restart or refresh of the server.

If you have determined that you need to refresh the Application Server, there are several ways to proceed. The simplest way to refresh the server is to stop and then restart the Application Server via the Services Control Panel. However, this requires manual intervention, and so would not be suitable for programmatic refresh (after an extraction, for example). This method also interrupts anyone currently using the system.

Another way to refresh the server is by using the **refresh.exe** program that was installed in your *installdir/win32* directory. This program sends a message to the Application Server instructing it to re-read all of the necessary information

Refreshing the Application Server

mentioned above. It also re-parses all of the template files (in case the TemplateDir Registry key was changed) and re-initializes the Security objects.

Refresh will not interrupt the requests that are currently running on the Application Server. After the refresh has been completed, however, users who were previously logged on will need to log in again. This is because the security restrictions that affect all users' rights have been changed.

You may invoke the Refresh program via a DOS command-line or open the RefreshApp dialog box and enter command options (see *Invoking RefreshApp* for instructions).

The Refresh Command Line

The Refresh command syntax is:

```
refresh localhost port username password
```

where:

<i>localhost</i>	Specifies the name of the machine on which the Application Server is running.
<i>port</i>	Specifies the port number on which the Application Server is listening.
<i>Username and password</i>	<p>Specify a valid NT account that has access to the Epiphany Application Suite.</p> <p>These are the same values required for an end user to log into the top-level Epiphany Web page. In general, any account that is defined in the Security folder of the EpiCenter Manager application will work. See <i>Security</i> on page 109 for more information.</p>

Epiphany Application Server

After the Refresh application has established a connection to the Application Server and sent the appropriate messages, it displays the following message and waits for a response:

Sent the REFRESH instruction to localhost

Normally, the Application Server takes about 30 seconds to refresh before returning an acknowledgment. (Times may vary based on the size of your EpiMeta database and the speed of your network, and other considerations.) Upon receiving this acknowledgment, the Refresh program displays:

```
Refresh [Build 3.2.0.1035]
-----
Connecting to localhost:8081
Sent the REFRESH instruction to localhost
The refresh SUCCEEDED.
REFRESH operation took 18827 ms.
```

Otherwise, the program indicates that it has failed by outputting the response that it does receive, or by indicating that it has failed. For example, here is the output of a negative interaction in which the user/password failed.

```
Refresh [Build 3.2.0.1035]
-----
Connecting to localhost:8081
Sent the REFRESH instruction to localhost
The refresh FAILED because the username/password combination was
invalid.
REFRESH operation took 3365 ms.
```

(The Refresh program always displays the amount of time that it has taken.)

Invoking RefreshApp

RefreshApp is the graphical user interface (GUI) version of **refresh.exe**. You may keep the RefreshApp window open and on your desktop during debugging.

Follow these steps to use the Refresh application:

1. Choose the Refresh application from the *Start* menu:
Programs\Epiphany\RefreshApp.
2. For the AppServer name, enter the name of the machine on which the Application Server is running.
3. For the AppServer Port, enter the port number on which the Application Server is listening.
4. For the username and password, specify a valid NT account that has access to the Epiphany Application Suite.
These are the same values required for an end user to log into the top-level Epiphany Web page. In general, any account that is defined in the Security folder of the EpiCenter Manager application will work. See *Security* on page 109 for more information.
5. Click Refresh to start the application.
Messages display in the bottom of the dialog box that tell you whether the Refresh succeeded or failed (and the time that this event occurred).
6. In case of failure, click View Log to determine what caused the failure. The information that is normally printed to the screen during the execution of **refresh.exe** is displayed in this window.

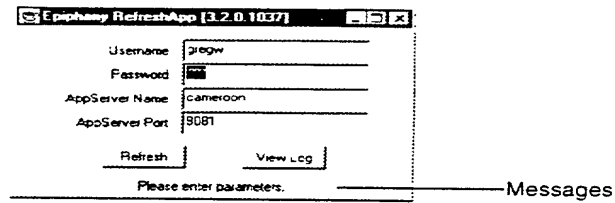


Figure 5-3 Epiphany RefreshApp Dialog Box

The EpiAppService Program

The EpiAppService program is the program that Windows NT uses to run the Application Server as an NT Service. You will use EpiAppService to—

- delete an Epiphany Application Server service.
- change the parameters of an Epiphany Application Server service.
- add a new Epiphany Application Server service; for example, create a service because the installation failed, or to install another instance.

The EpiAppService Program

Command Syntax

The **EpiAppService** command syntax is:

```
epiappservice -S servicename action
```

where:

-C Creates a service. Registers a new service with the NT system. You must use the -E option with this option to specify the executable to use as the NT Service; for example:

```
EpiAppService -S "instance" -C -E "program name"
```

When you are configuring an Epiphany Application Server, the command line will look like:

```
EpiAppService -S "instance" -C -E "C:\Program Files\  
Epiphany\instance\win32\EpiAppService.exe -S instance"
```

If the instance name has spaces in it, then you should use single quotes inside of the outside quotes. Do not use spaces within the instance name.

Run the EpiAppService program to create the service, which is also named EpiAppService. It is invoked with the -S option to specify the instance name. (The EpiAppService program is both the program that should be run as a service, as well as the program used to register the service.)

-E *cmd* The command to associate with the service.

Epiphany Application Server

- 09625518.072500
- DEP *service* Allows you to specify a service on which the Application Server depends. The Service Manager will always load this dependency before loading the Epiphany Application Server. The service name supplied as an argument to this option must exactly match the service name used by the service in question, usually the Microsoft SQLServer.
 - D Deletes the service specified via the -S option from the NT Service Registry. When this service is deleted, only the entry in the NT Registry for the service is deleted; no executables are deleted. Use this to remove unused Application Server instances.
 - G The launch handler for the service. (Go.)
 - T Starts a service. The installation program uses this option to start the Web server (which must be shut down during installation). It starts the service specified via the -S option, which is equivalent to clicking the Start button in the Control Panel\Services dialog box to start the desired service. You may use this option when you cannot access the Control Panels (for example, during RCMD or pcAnywhere remote access).
 - P Stops the service specified via the -S option. (See the -T option.) The installation program uses this option to stop the Web server during the installation.
 - K Checks if the service specified via the -S option is running. The program returns a zero return code if the service is running, and a non-zero return code if it is not.

The EpiAppService Program

- ? Displays online help, a description of the command options.

EpiAppService Command Examples

Examples of the most common invocation of this program follow:

- Creation of a service because the installation failed, or you wish to install another instance. If the database server is on a different machine from the Application Server:

```
EpiAppService -S instname -C -E "C:\Program Files\  
Epiphany\instname\Win32\EpiAppService -S instname"
```

If the database server is on the same machine as the Application Server:

```
EpiAppService -S instname -C -E "C:\Program Files\  
Epiphany\instname\Win32\EpiAppService -S instname" -DEP  
MSSQLServer
```

- Deletion of an old service.

```
EpiAppService -S instname -D
```

The Application Server's Registry Keys

All Registry keys used by the Epiphany Application Server are located in

HKEY_LOCAL_MACHINE/Software/Epiphany/Instances/instance_name

where *instance_name* specifies the name of the instance you entered during the Epiphany software installation. This directory contains the following keys:

AppServerHost	The name of the machine on which the Application Server is running. The default is the <i>localhost</i> .
Application ServerPort	The port number on which the Application Server is listening for connections. The default is 8081.

AppServerLogVerbosity This parameter specifies the degree of logging (verbosity level) that the Application Server performs. The default value is 0. (For this release only the default value is supported.)

AppServerQueryTimeout

The number of seconds before the Application Server automatically terminates long-running queries. A query that requires more than this number of seconds to process will be terminated in order to prevent the exhaustion of system resources and run-away queries from bringing down the server.

AppSessionTimeout

The value in seconds for the lifetime of an idle session. If a session has been idle for this many seconds, then it is removed from the cache.

Build

The version number of the Application Server. This value should be 3.2.

ChartsOutputDir

The directory (full path) in which the *.epc chart files will be stored.

DatabaseName

The name of the EpiMeta database.

DatabaseServer

The name of the database server on which the EpiMeta database is located.

DatabaseUsername

A username for logging into both the EpiMeta and EpiMart databases.

DatabasePassword

The password that corresponds to the DatabaseUsername account.

DatabaseType

Specifies the type of the database. This value should be *SQLServer6.5*.

The EpiAppService Program

Description	A textual description of the instance.
DSN	The name of the DSN that is used to connect to the EpiMeta database.
InstanceRootDir	The root directory into which the Epiphany Application Server has been installed.
ProxyLogFile	(Optional) Enables proxy logging. For example, setting <i>HKEY_LOCAL_MACHINE\SOFTWARE\Epiphany\Instances\Stromboli\ProxyLogFile</i> to <i>C:\proxy.log</i> will create a <i>proxy.log</i> file if it does not already exist, and log information for every proxy submit to the Application Server. If an invalid path is specified in this ProxyLogFile Registry key, no logging will be performed.
SystemLogDir	The directory in which the Epiphany Application Server log files are written.
SystemLogDirWebpath	The name appended to <i>http://machinename/instance_name/</i> that informs the IIS Web server of the log files' locations.
TempDirGarbageLifetime	The lifetime in seconds of a temporary or log file created by the Epiphany Application Server. After this specified number of seconds from the creation date, a temporary file, log file, or *.epc chart file will be erased by the Temporary FileManager in the Application Server. Garbage collection occurs at Application Server startup and periodically when there are free cycles.

Epiphany Application Server

TemplateDir	The directory in which the Epiphany Application Server templates are stored.
Driver	The JDBC driver used to connect to an EpiMart database. The default is connect.microsoft.MicrosoftDriver . <i>Do not change this value.</i>
SecurityClass	(Optional) The Java class to use for security authentication. If it is not specified, the com.epiphany.security.EpiNTLogon class is used.

The Epiphany Proxy

The Epiphany proxy (**Epiphany.dll**) is an ISAPI application that mediates the requests and responses between the IIS Web server and the Epiphany Application Server. All user requests for Epiphany pages are directed to the proxy `http://machinename/scripts/instance_name/Epiphany.dll`.

This proxy bundles the request into a package that conforms to a strict Epiphany format and sends the package to the Epiphany Application server through a TCP/IP socket. The proxy uses the Windows Registry to find the Epiphany Application Server. In particular, the proxy parses the *instance_name* out of the requesting URL. It then opens the AppServerHost and AppServerPort with that instance's Registry tree, and uses this information to connect to the Application Server. The Application Server processes the request and sends a result back to the proxy. The proxy displays the result in the user's browser.

Note: The *instance_name* in the URL must match the *instance_name* as defined in the Windows Registry (configured from the installation program). This allows one proxy to direct requests to several different Application Server instances. This is of value when you want to have two versions, such as a release and a test instance.

Proxy Logging

The **Epiphany.dll** ISAPI proxy supports rudimentary logging. If you suspect that the proxy is not passing all parameters to the Application Server, you can enable logging of all parameters that the proxy submits.

Proxy logging, which is optional, is a diagnostic tool for identifying problems, not a run-time logging facility. Because the log file may grow arbitrarily large if proxy logging is always enabled, use it only in the event of a suspected problem with the **Epiphany.dll** proxy.

An example of a proxy log file:

```
Processing new request. Header: mGET q p v0.8 oHTTP/1.0 nzhenya
P80 r192.0.0.147 aMozilla/4.04 [en] (WinNT; I ;Nav) u U/scripts/
stromboli/Epiphany.dll S/scripts/stromboli/Epiphany.dll Requested
dispatched. Request data length was 0

Processing new request. Header: mPOST q p v0.8 oHTTP/1.0
nzhenya P80 r192.0.0.147 aMozilla/4.04 [en] (WinNT; I ;Nav) u U/
scripts/stromboli/Epiphany.dll S/scripts/stromboli/Epiphany.dll
sEP7USER=emile&EP_PWD=zhenya&EP_TMPL=toplevel&EP_APPLICATION_TYPE=Clarity
Requested dispatched. Request data length was 72
```

The log file consists of blocks, with each block representing a submit to the proxy DLL. Each new submit starts with `Processing new request.` Header: and is followed by the header. The header consists of header items, each beginning with a letter that describes the type of item, followed by the value of item.

The first letter of the header item is defined below.

m	method POST or GET
q	query string (used with GET)
v	version
o	HTTP protocol
n	server name
P	server port
r	remote IP address
a	user agent
u	user name (if Web server is using authentication)
U	URL
S	script name

The header will be followed by the data section if the submit was of the type POST. First, a number of bytes is printed (172), then the actual data, which should be the same size as the number of bytes printed. Finally, if the submit was successful, the following line is printed:

Requested dispatched. Request data

If the submit failed, then the log file should read:

FAILED TO DISPATCH REQUEST DUE TO A SOCKET ERROR.

Application Server Logging

Most components of the Application Server maintain a log file of their activities. This section describes each of these logs, their directory location, and their diagnostic use.

The components that generate logs and the contents of those logs are briefly described below (and in more detail later in this section).

Server	The com.epiphany.server.Server class generates a log file of connections made to the Application Server. This file contains the time at which a connection was accepted, the number of that connection, the time at which the request was finished, the total time required by the request, and certain messages printed to the log during the processing of that request. The latter category includes the session ID of the connection, the template used to process that request, the number of bytes generated in the response, exceptions that might be generated during the processing of the request, or the user name used to log into the server.
SecurityManager	The com.epiphany.security.SecurityManager class generates a log that contains the users and groups in the system.
SaveRestore Manager	This creates a log that contains all of the SQL statements executed for Save/Restore purposes. Calls to certain functions such as getReportKey() and loadSaveRestoreQuery() are also logged with their return values.

09625518.072500

Clarity or Relevance Query Log

Each Clarity or Relevance request generates a log that contains the submitted ticksheet's parameters, information about how the ticksheet was parsed, all of the SQL statements executed during the processing of the request, and information about how long the processing took.

Log File Location

In the Windows NT Registry on the Application Server machine, there is a Registry key named *HKEY_LOCAL_MACHINE/Software/Epiphany/Instances/instance_name/SystemLogDir* that specifies the location of all log files generated by the Application Server.

Log File Naming Conventions

All log file names begin with a prefix that indicates the date and time when the log file was first created. The format is

YYYY-MM-DD_HH-MM-SS-MS*name*.txt

where YYYY stands for the year, MM for the month, DD for the day, HH for the hour, MM for the minute, SS for the second and MS for the millisecond. *name* can be one of the following: SRV, SECURITY, SAVE_RESTORE, or QM_sessionid. The logs of queries for Clarity and Relevance applications have the suffix QM_sessionid.

The Server Log

When you start the Application Server, the server object creates a log immediately after the Registry has been opened and read. The Registry must be opened first because it contains the *SYSTEMLOGDIR* Registry key that specifies exactly where the log files are stored.

```
**** Epiphany AppServer Acme awaiting connections... ****
Mon Jun 08 14:48:50 PDT 1998: +++++ Accepted [1] @ [897342530272]
----- Dispatched [1] @ [897342530322]
Free/Total Memory: 165064/3297272
Setting to default template (no app): toplevel
No session exists/created. Displayed login screen.
4865 bytes generated.
----- Finished [1] @ [897342530773]
501ms

Mon Jun 08 14:48:59 PDT 1998: +++++ Accepted [2] @ [897342539215]
----- Dispatched [2] @ [897342539245]
Free/Total Memory: 719832/3342328
Template passed in: toplevel
Got EP_USER: lslater
Adding session: EPIPHANY\lslater|192.0.0.12
5184 bytes generated.
----- Finished [2] @ [897342539615]
400ms
```

Note that the server writes this information for each socket connection it accepts:

```
Date: +++++ Accepted [connection number] @ [milliseconds since epoch]
```

For example:

```
Sun May 03 17:54:16 PDT 1998: +++++ Accepting [1] @
[894243256100]
```

The *Date* is written using the `java.lang.Date.toString()` method. Hence, it uses the default formatting on the server machine. The connection number is the

numerical ID of the connection that was accepted. Numerical IDs are assigned in increasing order, starting from 1. The *time in ms since epoch* is determined using the `System.currentTimeMillis()` method. It represents the number of milliseconds that have elapsed since January 1, 1970.

During the processing of the request, certain messages might be printed to the server log if no other log is available. This is the case for the TemplateServer and the EpiAdminServer.

After the request has been started, the server indicates the end time of the request in this format:

```
Finished [<connection number>] @ [milliseconds since epoch] 400ms
```

The Security Manager

The Security log first prints out all of the groups recognized by the system. Each entry in this first section contains the group name, the key in the database, and a list of ticksheets accessible by members of the group. For example:

```
Groups: {Administrators=
Group: Administrators / key: 4 / world save query access: 1
Access List:
Assigned ticksheets:
ExecutiveBestWorst, ExecutiveProfiles, ExecutiveProjections,
ExecutiveTrends, Expense,
```

The next section of this log file contains a list of all the users recognized by the system. Each entry contains a user name, the groups to which the user belongs, and the list of ticksheets that the user has permission to view. Note that the ticksheet list is mostly empty since all users belong to at least one group, and therefore inherit ticksheet permissions.

Application Server Logging

Save and Restore Manager

The Save and Restore Manager logs function calls made to **getReportKey()** and **loadSaveRestoreKey()**. All of the SQL executed during the retrieval/storage of a query is also logged to this file. Each entry is prefixed by the date and time of the entry. For example:

```
Sun May 03 18:55:26 PDT 1998: loadSaveRestoreQuery(Sales
Projections, sarah)
```

Clarity and Relevance Applications

Note: If the **log** link at the bottom of an HTML Clarity/Relevance result page does not take you to a log, make sure that the SytemLogWebDir *Registry* key refers to an alias that has been configured on the Web server. Also make sure that the directory and files have correct read permissions.

Each Clarity or Relevance application creates its own log file before performing any non-trivial processing. The log contains the version of the Application Server and the date/time of the request. The second line begins with APPLICATION parameters: and subsequent lines contain all of the name-value pairs submitted to the Application Server for this request. Multiple values submitted with the same name are shown separated by commas. A blank line follows, and then the Application filters, dimensions, and measures that could be parsed from the submitted ticksheet's parameters are logged.

Next, the system logs all of the SQL statements needed to compute the answer to the request, the amount of time in milliseconds to process the query, and the number of rows returned for all SQL queries.

The log ends with the CLEANUP message.

Application Server Security

Authentication is performed outside of the Epiphany system, which does not capture password information. This external authentication requires an authentication module. For example, the NT authentication module authenticates users with an NT domain controller. The Security Manager inside the Application Server loads the authentication module specified through the SecurityClass Registry key at initialization. Each authentication module supports a fixed API that includes methods to authenticate users, add new users to the Epiphany system, and sync-up outside information on the user (such as group memberships) with Epiphany metadata.

Currently, Epiphany provides two authentication modules: the NT authentication module EpiNTLogon, and an insecure authentication module called EpiPassThruLogon. Use EpiPassThruLogon only for testing and debugging. Optionally, you can add an LDAP authentication module, X.500 module, and other similar modules.

The optional Registry key SecurityClass under the *instance_name* Registry directory controls which security module is loaded. You must specify the full class path to the security module. If this key is omitted, the system uses the default security module `com.epiphany.security.EpiNTLogon`, which uses NT to perform user authentication.

The means by which the authentication information (username and password) reaches the Application Server is as follows. Users log into the Epiphany system either through a login template or through a Web server authentication mechanism, such as Basic Authentication or NTLM.

Application Server Security

When the Application Server receives the user name and password, it calls the Security Manager to log in the user. The Security Manager loads an authentication module, and attempts the login process. If the process is successful, depending on the authentication module, one of the following steps is taken:

- If the user exists in the EpiMeta database, user group memberships outside of the Epiphany system will be synched up with group memberships in the EpiMeta database.
- If the user does *not* exist in the EpiMeta database, but is authorized to use the Epiphany system, then a user account will be created in the EpiMeta database. User group memberships outside of the Epiphany system, such as NT, will be synced-up with group memberships in the EpiMeta database. For example, assume your EpiMeta had a group named EPIPHANYUSERS configured with access to all of the ticksheets. If a user name Joe (a valid NT user who belongs to the NT group EPIPHANYUSERS) supplies a correct password, then Joe will be automatically added to the EpiMeta metadata as a user who belongs to the Epiphany group EPIPHANYUSERS.

Only group memberships for a group whose Group Definitions dialog box in EpiCenter Manager has the Synchronize option selected will be synched-up. Sync-up occurs if:

- the user who logs in is 1) a member of a Group X outside of the Epiphany system; 2) Group X exists inside the Epiphany system; and 3) the user is not a member of Group X inside the Epiphany system. Sync-up will create a group membership to Group X for this user inside the Epiphany system.
- the user who logs in is 1) a member of a Group X inside the Epiphany system, but 2) the user is not a member of this group outside of the Epiphany system. Sync-up will remove a group membership from the Epiphany system.

If the sync-up process requires a creation of a new group membership for a user, certain access rights are set on this membership. The ability to save queries has the access rights of Save Group/Default, and global-level save access is Inherit. (See *Security* on page 109 for more information.)

For a group to be the same in EpiMeta and the NT domain, it needs to have the identical name (case sensitive) in both. The group name in the NT domain includes the domain name, such as *Epiphany\Sarah*. You need to make sure that the group name in the EpiMeta includes the domain name for that group.

When the user is authenticated and user information is synced-up, Security Manager determines if the user is authorized to use the Epiphany system. The user is so authorized if he or she belongs to at least one group in the Epiphany system. (The user must also must log in with a valid password.)

Important: An authenticated user is authorized to use the Epiphany system if and only if that user is a member of at least one group in the Epiphany system after sync-up.

Authentication Modules

The following authentication modules are included with the Epiphany software release 3.2.

Module	Class Name
NT (default)	com.epiphany.security.EpiNTLogon
Pass through	com.epiphany.security.EpiPassThruLogon

- NT (default)

NT domain authentication. NT groups are imported into the Epiphany system through EpiCenter Manager. As mentioned, these groups need have the Synchronize option selected in the Group Definition dialog box, which means that memberships to those groups will be synced-up. Users

Application Server Security

who belong to those groups can log into the Epiphany system. When a user logs in, his or her NT group memberships are synced-up to the EpiMeta database. It is also possible to create Epiphany-only groups inside EpiCenter Manager by simply not selecting the Synchronize option. Thus, the Epiphany administrator may create arbitrarily complex permission hierarchies using EpiCenter Manager, independent of the manner in which NT domain security is set up.

- *Pass through (for in-house debugging purposes only; should never be used at a customer site after the initial Epiphany system setup)*

This is an insecure authentication that has no password. This is an Epiphany-only development module that ignores NT authentication all together. You do not need passwords in this model: specify your user name exactly as it appears in EpiCenter Manager (it must include a domain name if such exists), and you will be logged in. There is no sync-up process. That means that an authenticated user that does not exist in the EpiMeta database will not be allowed to use the Epiphany system. Epiphany groups and users are created and managed through EpiCenter Manager.

Authentication Module Tips

- If you want to use NT-related authentication modules, make sure that **EpiNTLoginJNI.dll** is in your system path.
- Users that do not belong to any groups in the Epiphany system are not allowed to log in. An error message that says that user is authenticated but not authorized to use Epiphany system is displayed whenever an unauthorized but NT-authenticated user logs into the Application Server. User memberships may be adjusted upon login if the user is a member of synchronized groups in Epiphany system. A user must be a member of at least one Epiphany group after the synchronization process completes.

- The optional Registry key `SecurityClass` (located in the *instance_name* Registry directory) controls which security authentication module is loaded. The full class path to the security module must be specified as the value for this key. If this key is omitted, the default security module is **com.epiphany.security.EpiNTLogon**, which uses NT to perform user authentication.
- The **Toplevel.template** is the template that appears immediately after a user logs into the Epiphany system. The name of this template is configurable with the an optional `ToplevelTemplate` Registry entry in the *Instances* directory. You can load the **company.template** instead of **toplevel.template** by substituting your company name in `ToplevelTemplate=company`.
- Depending on how IIS security is set up, one of the following situations occurs upon login:
 - If Allow Anonymous authentication is enabled, the login dialog box is displayed when the user attempts to use the system for the first time, or the user session times out. It is almost always sufficient to specify the username only. The domain name is located automatically.
 - The search order that authentication uses to find the user account is as follows. First, the authentication mechanism looks in the local machine's Security Access Manager (SAM), then it checks with the primary domain controller, and afterwards checks with trusted domains. If there are multiple users with the same name between domains and/or a local machine that runs Application Server, specify the full user *name-domainname\username*, or the *local_machine_name\username* if the user logs in from a local machine's Security Access Manager (SAM).
 - If Basic Authentication is enabled, the browser displays a login dialog box when the user attempts to access the Epiphany

system for the first time. However, when the user's session times out, no re-login is required. The user is automatically logged in again, and a new user session created.

- If NTLM authentication is enabled, Internet Explorer automatically performs authentication of the user without displaying a login dialog box, although Netscape displays it. If Basic Authentication or NTLM is on, the login dialog box does not appear in the Web browser.

Important: Do not create your own domains. Doing so introduces multiple domains, with the Epiphany machine in one domain and the user accounts of the Epiphany system in another domain. In order for the authentication module **com.epiphany.security.EpiNTLogon** to work properly, a two-way domain trust between the Epiphany domain and the customer domain is required.

If you set up a new domain for the machine that runs the Epiphany Application Server, set up a two-way trust and name the machine and the domain differently. In general, do not use the same string for domain names, machine names, and user names.

- The following applies to Synchronized groups:

EpiNTLogon requires the NT domain for lists of global and local groups. The names of these groups will be matched to the names of the groups in the EpiMeta. A group name will be matched if its domain name and group name match. Therefore, group *foo* will not match to group *EPIPHANY\foo*. (The match is case insensitive for both group name and domain name.) If a user is a member of an NT group *EPIPHANY\foo* and EpiMeta has a group called *epiphany\FOO*, there will be a match.

- If a user is a member of a local group and the group has a global group as a member, the global group will not be picked for synchronization process. Only the groups for which the user is an immediate member are considered for synchronization process.

- If a user logs in with an account that is local to the Application Server machine, then a membership to a special group called None is automatically retrieved from the machine's SAM. (Every user in the local SAM has a membership in a special global group called None although this group does not exist on the machine.) For this reason, do not create synchronizable groups called None in EpiCenter Manager.
- Avoid having the same name for domain names, machine names, and user names. For example, if the Application Server runs on the machine *foo*, and the user called *foo* attempts to log in, access may be denied. If the Application Server is running on a machine named *foo*, and a user named *foo* logs in from the primary domain, or from the local Security Access Manager (SAM) of the machine *foo*, authentication will succeed. If user *foo* logs in from a trust domain, however, the authentication will fail. The only way to log in as *foo* from another domain is to give the full name for the user account upon login: *domainname\foo*.

Configuring Output Templates

Application Server output results from a template-processing mechanism. Template files are files parsed by the system and used as templates for the output that the Application Server produces. For example, all of the static HTML and JavaScript that is a part of every Epiphany-produced ticksheet resides in a template file.

Configuring Output Templates

A template file contains three different kinds of text: plain HTML, a special environment variable tag, and a special class invocation tag. These are described below.

Plain HTML

Most of the template file contains plain text, Java script code, or HTML that is copied verbatim when the output is generated. This allows a system administrator to make small modifications to the look and feel of certain pages. For example, one can change the **clientlogo.gif** that appears in the upper right-hand portion of the Web page.

A special environment variable tag

The `<!--EP ENV="{name}"-->` tag is an environment variable tag that is parsed and textually substituted with a string from the environment hash. The Environment hash is a structure that is maintained by the Application Server during the processing of a query. It contains name-value pairs meant to be accessible from a template file. When this special tag is parsed, {name} is used to query the Environment hash for a value string. The string is then printed in place of the tag. The tag can contain special instructions for the encoding of the value string. For example `<!--EP ENV="foo" QUOTEESCAPE -->` forces the Application Server to encode all of the `<` and quotes in the value string as `>`; and `"`; . There is also a `URLESCAPE` that encodes the value string as per the x-www-form-urlencoded MIME type. See the online MIME standards RFC 2045 through RFC 2049 for more information. The URL is www.oac.uci.edu/indiv/ehood/MIME/MIME.html.

A special class invocation tag

The class invocation tag

```
<!--ENV CLASS="com.epiphany.presentation..."-->
```

instructs the Application Server to load dynamically the class specified as the CLASS parameter, and to invoke the *execute* method of that class. Any text that is produced by that class is substituted in place of the tag. Most of the dynamic content of an HTML result page is generated by a class invocation. For example, the table of results on a Clarity result page is generated by the class:

com.epiphany.presentation.HTMLTableOutputMethod

To support localization, class invocation tags also support two auxiliary parameters, LANGUAGE and COUNTRY. These parameters localize the text that is produced by the class.

For example, in all of the templates shipped with the installation program, you will find the following method at the bottom of the page. This method displays the date and time at which the output was produced.

```
<!--EP CLASS="com.epiphany.presentation.DateMethod"
LANGUAGE="en"-->
```

Although you may make simple HTML changes to the template files, do not change the environment variables (see *Environment Variables* below), or the class invocation tags. The environment variables allow you to substitute installation-specific or query-specific information into the result pages. For example, the name of the machine on which the Application Server is running can be accessed by the EPV_URL_PREFIX environment variable. This information is required for any hyperlink or drill-down request that requires communication with the Application Server to work. The alternative would be to hard-code the machine name into templates. This, however, requires tedious

Configuring Output Templates

and error-prone editing to be done at every installation site and every time that a machine name changes. In some cases, it is impossible to hard-code a fixed value into the template.

For example, after a user logs in, he or she is assigned a unique session ID that is embedded in each of the ticksheets that the user sees. The session ID is generated randomly on the server at the time the user logs in, so there is no way to hard-code this information into the template.”

Environment Variables

The environment variables available for customer use are described below.

APPSRVR_QUERY_TIMEOUT

The same value as the Application ServerTimeout Registry key.

EPA_ANCHOR

Used when someone clicks a hyperlinked item in a ticksheet to display its dictionary entry.

EPV_APPSERVER_VERSION_STRING

The version string for the Application Server, for example, 3 . 2 . 1 0 2 9.

EPV_BESTWORST_ATTS

Used in Relevance Best & Worst ticksheets as a list of attributes.

EPV_INSTANCE_NAME

The name of the instance.

EPV_URL_PREFIX

The URL of the machine. This value is parsed from the initial URL that is used to access the Application Server.

EP_SES

The session ID of the current session. Used in place of the *username/password* after a user has logged in.

EP_TMPL	The template that will be used to format the output.
EP_USER	The username of the user logging in. This value is only available when the user initially logs in.
EXTENDON_SELECT	A special purpose parameter used in the profiling machinery.
QUERYLOGFILENAME	The name of the query log file. It is defined for all Clarity and Relevance queries.
ERRORMESSAGE	When the system encounters an error, this variable contains the text that describes the error condition.

09625518.072500

Customizing Error Messages

You may customize most of the Application Server's error messages via the file *errormessages_en.properties* located in the *installdir\Classes* directory, where *installdir* is the directory in which you installed the Application Server. The prefix *_en* indicates that these are English-language messages. In future releases, the Application Server will support error messages in multiple languages.

The text that displays in the Application Server log when the server receives an exception corresponds to the text entered in this file for each anticipated exception.

Every unique exception generated by the Application Server has a name-value pair line in the *errormessage_en.properties* file. For example:

```
>-----  
  
com.epiphany.exception.EpicenterInvalidValueException=Invalid value  
found in the metadata: {0}.\nWhile executing the following sql  
statement, the Epicenter loader encountered an invalid value. Please  
make sure that the epicenter has been built properly, and that it has  
not been corrupted.\nSQL Statement:\n{1}  
  
com.epiphany.exception.EpiQueryInvalidMeasureException=You have  
selected a measure that is not defined in the metadata for this  
ticksheet.\n{0,choice, 1#The Measure you selected was|1< The Measures  
you selected were}:\n{1}\nGo back to the ticksheet and choose another  
measure.  
  
>-----
```

The name-value pair appears on one line. If your messages are cut off, make sure that there are no line breaks between the text. If you need to insert new lines in your message text, use `\n`.

Briefly, each exception has a function that makes the appropriate calls to read the exception text from the *errormessage_en.properties* file and to process it by substituting the correct strings. The macro expansion capability is simple. The parameter {n} refers to the nth argument passed to the exception. The parameter

```
{n, choice, i#blah blah|k< blek blek}
```

indicates to print either *blah blah* or *blek blek* based on the nth argument to the exception.

- If *n==i*, then it is *blah blah*.
- If *k<n*, then it is *blek blek*.

(You cannot include brackets ({ }) in the exception text.)

This architecture—

- separates the text describing the error from the state of error.
- facilitates localization of error messages.
- allows you to change the error text without having to recompile.

Name Replacement

The file *dictionary_en.properties*, located in the *installdir\Classes* directory, allows the system administrator to replace dynamically occurrences of words with any arbitrary sequence of words.

The structure of the *dictionary_en.properties* file is exactly the same as the *errormessages_en.properties* file. The file consists of name=value pairs on single lines. For example, a file with the following line:

```
Fiscal Year=L'ann\u0431e fiscal
```

Customizing Error Messages

substitutes the string `L'année fiscal` for the string `Fiscal Year` in all of the output pages. The name=value pairs can contain escaped Unicode characters. The standard way to escape a Unicode character is to use `/uXXXX` where the `XXXX` represent the hexadecimal representation of a character's code. By default, no name replacement is performed. If the *dictionary_en.properties* file is not present, then the Application Server displays this status message for each query:

Dictionary for en not found.

To remove this message, create a dummy *dictionary_en.properties* file that contains no characters.

See Appendix I for a description of the Application Server error messages and conditions.

09065518 072500

Installation

This appendix covers the Epiphany software installation procedures.

Epiphany System Requirements

Before installing the Clarity and Relevance 3.2 software, please ensure that your system meets the following specifications:

- An Intel-based NT Server 4.0, Service Pack 3, IIS 3.0 (Web server), and Microsoft SQLServer 6.5 with Service Pack 3 or later.
- A minimum of 128 Mb of RAM; 256 or 512 Mb is preferred. The machine that runs the Epiphany Application Server requires a minimum of 64 Mb.
- One or two CPUs (two are preferred) with a speed of 200 or greater MHz.
- An adequate amount of disk space. The Epiphany-specific software requires about 8 Mb. You will also create two new, empty databases: one for EpiMeta (metadata) and the other for EpiMart (your data warehouse).

The recommended size for EpiMeta is 50 Mb. for data and 50 Mb. for logs.

Installation

The size you allocate for your EpiMart depends on the amount of your customer data. If you already have specifications for your EpiCenter and know the number of fact rows and dimension rows, you can use the following rough estimate in which the number of bytes of space equals:

```
2 * (10 * #fact rows * (100 + #bytes user-fact-data)
+ #dimension rows * (20 + #bytes user-dimension-data))
+ 300 Mb
```

This calculation is based on two copies (A vs. B) of the database and aggregates that are approximately ten times the base table size, with additional space added for staging tables, indexes, and keys. *Aggregates* are pre-calculated summaries stored in the data warehouse to accelerate queries.

As part of the Epiphany software installation, you will need to allocate at least 200 Mb for the temporary database *tempdb*, and 100 Mb for the log database.

- A RAID disk array is preferred although not necessary. RAID-0 is rated as having the highest performance; RAID-5 emphasizes data integrity.
- The computer must be connected to a network and have TCP/IP installed.
- Microsoft Exchange messaging client software must be installed for e-mail notification regarding the status of the database extraction process.

Notes:

When you install the Epiphany software, the first screen displays your system configuration.

Multiple network card configurations are not currently supported.

Installation Overview

The overall steps for installing the Clarity and Relevance 3.2 software involve—

- installing Microsoft Windows NT Server 4.0 (and its service pack; the service pack installs the version of the IIS server you need).
- installing Microsoft Access (one of the Microsoft Office programs).
- installing Microsoft SQL 6.5 Server (and its service pack).
- configuring Microsoft SQLServer.
- installing Clarity and Relevance 3.2 software from the Epiphany CD-ROM.
- setting up the Epiphany Application Server.

The Epiphany software does not have to be installed on the machine that has Microsoft SQLServer software installed. You also have the option of installing for remote administration.

Note: If applicable to your site, you will need to create custom DSNs (ODBC Data Source Names) that are required by connectors.

The complete installation procedure as documented here takes approximately two hours.

These instructions are directed towards consultants and database administrators who are familiar with installing software for the Windows NT operating system. If specific instructions are not given, the default values are acceptable.

If you have any questions about your network, please consult your local network system administrator. For third-party applications, please refer to the documentation that came with your software for more information.

Installing Windows NT Server 4.0

Follow these steps to install your Windows NT Server 4.0:

1. Insert Disc 1 of the Microsoft Back Office 2.5 CD-ROM into your CD-ROM drive, open the CD-ROM, and double-click the **Setup.exe** file.
2. The Welcome to NT Server Setup screen displays. Press Enter. Follow the instructions on the screen as they pertain to your system.
3. Select the directory location to install the Windows NT Server 4.0. (The default is acceptable.) The files are copied to your computer.
4. After the files have been copied, remove the CD-ROM and restart your computer as indicated.
5. When the Windows NT Setup screen appears, begin the Windows NT Setup.

Windows NT Setup

For the Windows NT Setup:

- Assign a computer name of 15 or fewer characters and write this name down for future use.
- Indicate the server type as a stand-alone server.
- Enter your administrator account name and password.
- Create a repair disk in the event of an emergency.
- Select components (the default values are acceptable).

Windows NT Networking

If your computer is part of a network, the Wired to Network field displays the network name and the Microsoft Internet Information (IIS) server (the Web server) is automatically installed. The network settings must be configured after consulting with your local network system administrator (see Step 4).

To set up Windows NT networking:

1. Select Start Search for network adapters.
2. TCP/IP as protocol is selected (and cannot be deselected).
3. Accept the default selections for Network Services and click Next to continue.
4. Depending on what your network system administrator specifies, either select DHCP or enter your TCP/IP address.
5. In the TCP/IP Properties dialog box, enter your TCP/IP address, the Subnet Mask, and Default Gateway (as assigned by your local network system administrator).

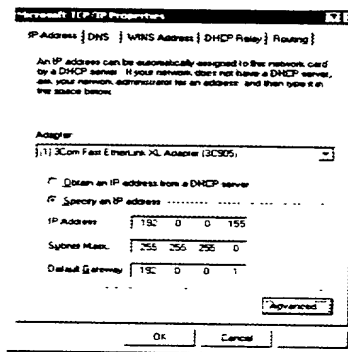


Figure A-1 TCP/IP Properties Dialog Box

- Click the dialog box's DNS tab and enter the domain name.
- Click the WINS Address tab. In this dialog box, enter the address for the primary WINS server. Click Apply in the WINS Address dialog box, and then click OK.

Installation

- For Show bindings, select all services from the check box list. Normally, the defaults are acceptable.
- Do not click Enable unless a red international symbol for NO (red circle with slash through it) appears next to a service. This will not appear on a clean install.
- Click Next.
- Start the network by clicking Next.
- In the Workgroup/Domain dialog box, enter the domain name in *Create computer account in the domain ONLY* if this is appropriate for your site.

This completes the Windows NT Setup. Next you will set up the Internet Information Services.

Microsoft Internet Information Server (IIS) Setup

Note: Windows NT Setup installs IIS 2.0; the Service Pack 3 upgrades IIS 2.0 to IIS 3.0.

As part of the Windows NT Setup, after you finish setting up the network parameters, the Finishing Setup dialog box is displayed, which informs you that you will now start setting up MS IIS 2.0. Follow these instructions:

- Use the default values for the Options dialog box.
- You may specify different values in the Database Publishing Directory dialog box. If the security features of IIS are to be used, the *wwwroot* directory must reside on an NTFS file system.
- Select the Microsoft SQLServer driver to be installed.
- Set the Time Zone.

This completes the Windows NT installation. You may remove the CD-ROM and restart the machine.

Note: After installing the new operating system, you may have to install drivers for special graphics cards or other hardware.

Install the Service Pack

The last step of the Windows NT installation procedure is to install the Microsoft Windows NT Service Pack 3. The Service Pack provides upgrades to Windows NT 4.0 and installs the IIS 3.0 Web Server, replacing the 2.0 version installed during the Windows NT installation.

Follow these steps to install the NT Service Pack:

1. Log on to Windows NT as the administrator.
2. Insert the Service Pack 3 CD-ROM into your CD-ROM drive.
3. Click Install Service Pack on the screen (or run **spsetup.exe** from the CD-ROM). The Service Pack Setup dialog box displays.
4. Because this is a clean install, you do not want to create an uninstall directory.
5. Click Finish, remove the CD-ROM, and restart your computer.

Install Microsoft Office

Microsoft Office may be installed from the Microsoft Office disk.

1. Insert this CD-ROM into your CD-ROM drive and follow the instructions on the screen.
2. Accept all of the defaults.

Installing Microsoft Office installs Microsoft Access, which installs the database used for importing and exporting metadata.

Installing Microsoft SQLServer

You will install SQLServer and its utilities from the Microsoft Back Office CD-ROM, Disc 2.

Follow these steps:

1. Insert the CD-ROM Disc 2 into your CD-ROM drive and open the disk on the desktop.
2. Open the Sql65 folder and then open the i386 folder. Double-click Setup.
3. Select Install SQLServer and Utilities.
4. For SQLServer's installation path, use the default—if there is enough space.
5. For the Master device creation, use the default—if there is enough space.
6. In the Installation Options dialog box (Figure A-3) click the Sets button next to Character Set, which displays the Select Character Set dialog box. Select 850 Multilingual (see Figure A-2) and click OK.

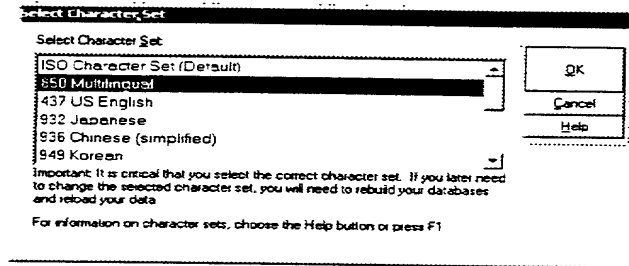


Figure A-2 Select Character Set Dialog Box

7. In the Installation Options dialog box, you may use the default Sort Order.

8. In this same dialog box, click the Networks button. For additional network support, select Named Pipes and TCP/IP Sockets in the Select Network Protocols dialog box (Figure A-3).
9. Select the check boxes Auto Start SQL Server at boot time and Auto Start SQL Executive at boot time (so these start whenever the Windows NT server boots).
10. After completing the Installation Options dialog box, click Continue.
11. When asked for the SQL Execution Log on Account, enter your password.
12. For the TCP/IP Socket Number, use the default port number.

Now you are ready to install the Service Pack.

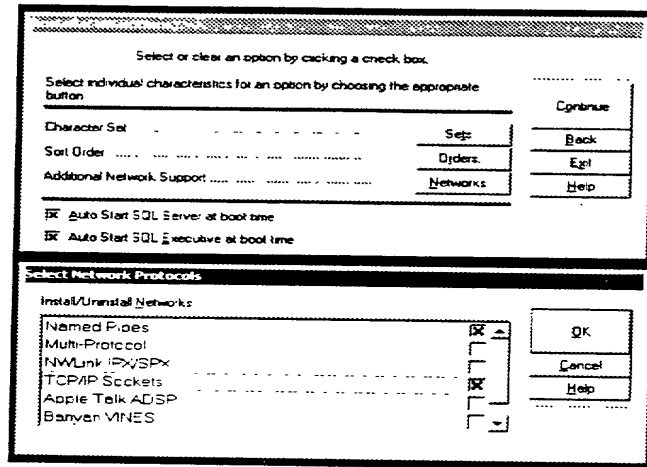


Figure A-3 Installation Options: Select Network Protocol

Installation

Install the Service Pack

Install the Microsoft SQLServer Service Pack 3 to upgrade SQLServer.

- Insert the CD-ROM into your CD-ROM drive and follow the instructions on the screen.
- After installation, remove the CD-ROM and restart your computer.

Microsoft SQLServer Setup

Follow these steps to set up SQLServer version 6.5:

1. Log in and start the SQL Enterprise Manager.
2. Select Server Register Server from the menu. Specify **sa** as the login ID and leave the password field blank.
3. Enter the machine name in the server box.
4. Click Register.

Setting Up Your Databases

Use the SQLServer Manager to set up new databases (see Figure A-5). You need to configure disk space and database space for two new, empty databases: one for your EpiMeta (metadata) and one for your EpiMart (data warehouse). The recommended size of your EpiMeta is 100 Mb; 50 Mb for data and 50 Mb for logs. (See *Epiphany System Requirements* on page 201.)

Right-click the Database Devices directory. Select New Device from the pop-up menu. The New Database Device dialog box (Figure A-4, page 211) is displayed.

To set up one of your two databases:

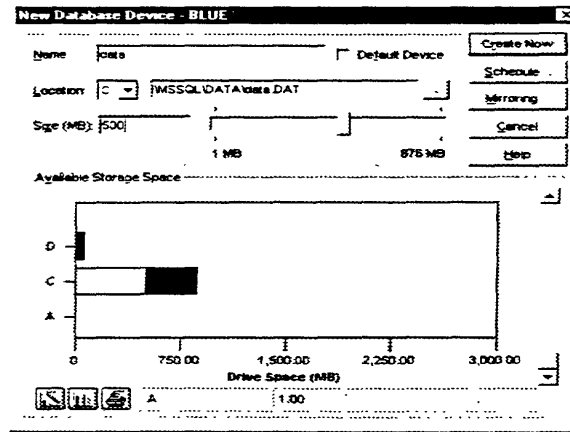


Figure A-4 New Database Device Dialog Box

1. Enter the name for the database device. The naming you chose for your EpiMeta should distinguish it from your EpiMart; for example, *Corp_EpiMeta* and *Corp_EpiMart*.
2. Select the location where there is enough disk space available and enter the size in megabytes. Click Create Now.

The size needs to be large enough to store the EpiMeta or EpiMart database for your site.

You may modify this size at any time.

3. Select *tempdb* in the database tree and right-click it. Select Edit from the pop-up menu. In the Edit dialog box, click Expand.

Installation

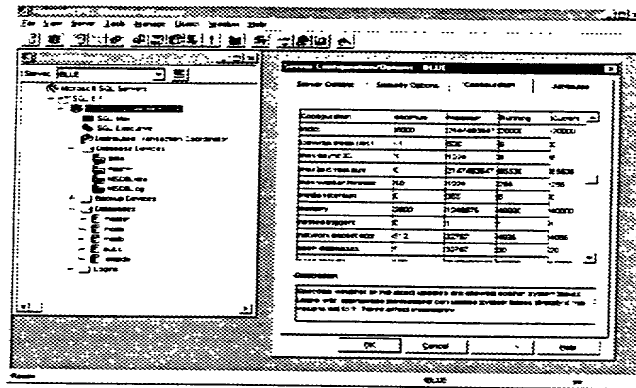
4. In the Expand dialog box, enter the size to expand the temporary database, in addition to the device where it is to be expanded (the recommended size is 200 Mb).
5. Click the Expand button. The *tempdb* will be expanded to the indicated size.

Repeat this procedure to create your other database.

SQLServer Configuration

Using the Server Manager, right-click the server you just installed in the Microsoft SQLServers tree; see Figure A-5. Select the Configuration tab in the Server Configuration/Options dialog box. Change the following values as indicated:

- *Locks*. Set the value to 20,000.
- *Memory*. Assign as much memory as your system configuration allows. For example, 100,000 2 Kb blocks equals 200 Mb of memory.
The value that is displayed is a theoretical maximum, *not* the maximum level that the machine will handle. If you set the memory value higher than a value your machine can run, SQLServer will not restart.
- *Open objects*. Set the value to 15,000.



Installing the Epiphany Software

Depending on your system configuration, you may be restricted to the Remote Administration setup. If so, the installation screen provides instructions.

- EPIPHANY CONFIDENTIAL*

Installation

important: The Microsoft Java virtual machine needs to be installed on your system. If it is not, you will be prompted to exit the installation program and install it. This software is included with Internet Explorer 4.0, which is available on your installation media (\\386\\Utils\\I.E 4.01).

To install the Epiphany software:

1. Exit all open Windows applications, and if this is a re-install, or an install over a previously installed system, make sure that the Epiphany Application Server has been stopped.
From the Start menu, go to *Settings\\Control Panel\\Services* and manually stop the Application Server. If you do not, the DLLs that are copied by the installation program will not be copied correctly.
2. Insert the Epiphany CD-ROM into your CD-ROM drive.
3. Double-click **Setup.exe** on the CD-ROM.
4. If the Microsoft Java virtual machine is not installed on your system, you are requested to exit this installation and install it.
5. The System Configuration screen displays your system's configuration data (for your information).
6. Select the type of setup: *Application Server* or *Remote Administration*. Follow the instructions below for your type.

Application Server

A Services window shows which services will be stopped before the file copy begins.

1. Enter the instance name. This is the name of the service as it appears in the Service Control Panel.
An instance name distinguishes among multiple installations of the Epiphany software on the same machine. Typically, you will install the software once on a machine and thus have one instance. You are asked if this instance will be the default instance. If so, the **Setup** program makes this instance the default Web server destination.
2. Enter the name of the database server and the database name. The database name is the name of the EpiMeta database. This database does not need to exist yet. (The Registry keys are populated based on what you enter.)
3. Enter the username/password for the database server. The user must have system administrator (SA) privileges.
4. Select the destination directory for the Epiphany software. The default is *C:\Program Files\Epiphany\instance_name*.
5. Enter the location of the run-time reports and charts (ticksheets) that end users will access. *C:\Program Files\Epiphany\instance_name\Charts* is the default. Create a new folder as requested.
6. Select the components to be installed: Epiphany System Programs, Applications Server Runtime files, Applications Server Java Classes, and on-line documentation in Adobe Acrobat/Reader PDF format.
If you want to install only the Applications Server, select Applications Server Runtime files and Applications Server Java Classes (and the documentation); otherwise, select all of these to be installed on your machine.
7. Select the default program icon location.

Installation

This completes the Epiphany software installation. Next, go to *Setting Up the Epiphany Application Server* on this page.

Remote Administration

1. Select the destination directory for the Epiphany software. The default is *C:\Program Files\Epiphany\instance_name*.
2. Enter the location of the run-time reports and charts (ticksheets) that end users will access. *C:\Program Files\Epiphany\instance_name\Charts* is the default. Create a new folder as requested.
3. Select the components to be installed: Epiphany System Programs and on-line documentation in Adobe Acrobat/Reader PDF format. For the Remote Administration, do not select the Application Server components.
4. Select the default program icon location.

This completes the Epiphany software installation.

Note: If you want to use the NT Performance Monitor to monitor the extraction process, go to page 222.

Setting Up the Epiphany Application Server

The Epiphany proxy is the only component of the Epiphany Application Suite that interacts with IIS. For the Epiphany Application Suite to function properly, set the IIS configuration values according to the instructions in this section.

The Epiphany Application Server requires sufficient memory (at least 64 Mb) to function properly. The machine that runs the Application Server needs to be set to 256 colors mode.

The steps you follow depend on whether you are using IIS 3.0 or IIS 4.0 (see *IIS 4.0 Settings* on page 219).

IIS 3.0 Settings

You will use the Internet Service Manager dialog box to configure these settings.

1. From the Start menu, select *Start\Programs\Microsoft Internet Server (Common)\Internet Service Manager*.
2. Right-click your server icon in the Internet Service Manager window. The WWW Services Properties dialog box is displayed.
3. Select the Directory Security tab. In the Anonymous Access and Authentication Control panel, click Edit.
4. The dialog box that displays has tabs for Services, Directories, Logging, and Advanced, which need to be set as described below.

Note: These are the default settings.

- Services Tab

TCP Port = 80

Connection Timeout = 90

Maximum Connections = 100000

Anonymous Login Panel values:

- Username

An NT username that has file access permission for all of the Epiphany installed files and directories. This username also needs permission to read the Epiphany entries in the Registry; that is, all entries under the *HKEY_LOCAL_MACHINE\Software\Epiphany* directory in the Registry.

- Password Authentication Panel values:

The password for this username.

Installation

Password Authentication

- Select Allow Anonymous Login.

- Directories

Select the directory name in the listing.

Click Edit Properties in the Edit Properties dialog box.

Select Virtual Directory.

Enter the alias name for these directories in the text box.

Directory	Alias
C:\inetpub\scripts	\scripts
C:\ProgramFiles\Epiphany\ instance_name\web\wwwroot	\instance_name

Allow read and execute access. Click OK.

Configure the following two fields:

- Select Enable Default Document. Enter this default document name: **default.htm**
- Select Directory Browsing Allowed.

- Logging Tab

You may use the default IIS settings, which are logging enabled, log to files, logging from standard, automatic open, log daily, log file directory: *C:\WINNT\System32\LogFiles*.

- Advanced Tab

Select Granted Access.

IIS 4.0 Settings

Note: For the Clarity and Relevance 3.2 release, Epiphany has not thoroughly tested IIS 4.0. Preliminary tests, however, have not indicated any compatibility problems.

To configure Applications Server settings for IIS 4.0:

1. Open the IIS 4.0 Service Manager from the *Start* menu by choosing:
Programs\Internet Service Manager.
2. Right-click your server icon and select Properties from the pop-up menu. The Properties dialog box is displayed in the Master Properties panel. Select WWW Services as the Master Properties. Click Edit. In the WWW Service Master Properties dialog box, select Directory Security. In the Anonymous Access and Authentication Control Panel, click Edit. The Authenticate from Methods dialog box is displayed. Select Allow Anonymous Access. Click Edit.
3. Configure Anonymous Login with file access permissions for reading and writing to all Epiphany files.

Anonymous User Account:

- Username
An NT username that has file access permission for all of the Epiphany installed files and directories. This username also needs permission to read the Epiphany entries in the Registry; that is, all entries under the *HKEY_LOCAL_MACHINE\Software\Epiphany* branch in the Registry.
Make sure that the directories that contain the **Epiphany.dll** and the **makechart.dll** files have execute permissions.
- Password
The password for this username.

Installation

4. Using the Virtual Directory tab, set up aliases for the following two directories:

Directory	Alias
C:\inetpub\scripts	\scripts
C:\ProgramFiles\Epiphany\instance_name\web\wwwroot	\instance_name

Manual Chart Install

The Epiphany charting solution consists of two modules: an ISAPI **DLLmakechart.dll** and a COM object **gsmaker.exe**. You need to install the charts on the machine that runs the AppServer. This machine needs to be running in 256-color mode.

Before proceeding with this installation, stop the Web server and kill the **gsmaker.exe** process if one is already running. Stopping the Web server is necessary to unload the **makechart.dll**. If you are unable to stop the Web server or kill the **gsmaker.exe** (using the Task Manager or **kill** utility), reboot your machine.

Place **makechart.dll** in the same directory as the *www* directory for the instance. If there are multiple instances present on the machine, you need **makechart.dll** in each instance *www* directory. (You can copy **makechart.dll** to the destination.)

Note that **makechart.dll** requires the following Registry entries under *HKEY_LOCAL_MACHINE\Software\Epiphany\Instances\instance_name*.

- ChartsOutputDir

The physical directory where the *.epc charting files are stored. These files are used to display charts, and are generated whenever charts are requested. By default, these files are removed daily by the Temporary File Manager. To set the number of seconds between Temporary File Manager runs, change the Registry key *TempDirGarbageLifetime*.

- SystemLogDir

The directory where the *charts.log* file is to be placed. This log file logs every call to the **makechart.dll**.

The **gsmaker** executable is a Microsoft COM object that needs to be registered. If **gsmaker.exe** was already installed on this machine, then make a backup of the old executable and copy a new **gsmaker.exe** over it. Since the old COM object is registered, you do not need to re-register the new one.

If, however, you are installing **gsmaker.exe** for the first time, you need to register it as follows:

1. Copy **newgsmaker.exe** to
C:\ProgramFiles\Epiphany\instance_name\win32, and change to that directory.
2. Run the following commands (which are case sensitive):
`gsmaker -UnregServer`
`gsmaker -RegServer`

The **gsmaker.exe** process appears in the Task Manager if the registration was successful.

To verify whether **gsmaker.exe** is registered properly:

1. Run **dcomcnfg.exe** from the command line.
2. Select Application EpiChart Class.
3. In the General tab, make sure that the **gsmaker.exe** entry points to the right location. In the Identity tab, make sure that Launching User is selected.
4. In the Security tab make sure that the Everyone special group has access and launch permissions to **gsmaker.exe**.

09625518.072500

Installation

If custom permissions are set, click the Edit button to check permissions. If default permissions are selected, close the EpiChart Class Properties dialog box and open the Default Security tab to check the permissions.

The **gsmaker** program uses the Graphics Server Charting Package to draw the charts. You need to add the following Graphics Server files to the *C:\winnt\system32* directory if they are not already present:

- **gsgif32.dll**
- **gsprop32.dll**
- **gsw32.exe**
- **gswag32.dll**
- **gswdll32.dll**

This completes the Application Server setup. If you want to use the NT Performance Monitor to monitor the extraction process, after setting up your Application Server, go to the next section.

Setting Up NT Performance Monitoring for Extraction (Optional)

Performance monitoring of the extraction process is optional. Although every **extract.exe** is instrumented using the standard NT Performance Monitoring facility, you need to take these steps to use the NT Performance Monitor to monitor an EpiChannel job's progress:

1. Your *Epiphany\instance_name\win32* installation directory must contain these items: **EpiPerfMon.dll**, **EpiPerfMon.ini**, and **EpiPerfMon.reg**.
2. **EpiPerfMon.reg** has values specified for this Registry key:
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Epi\Performance

Running **EpiPerfMon.reg** enters this key and its values into the NT Registry. Before doing so, you may edit this file so that the value *Library* points to the correct path where **EpiPerfMon.dll** resides, that is, your current *Win32* directory.

Edit **EpiPerfMon.reg** to change the *instance_name* to the correct value for the *Library* path, save the file and exit.

Note: Because most Epiphany files are installed as read-only files, you need to change permissions before editing.

3. Run **EpiPerfMon.reg**. To check the results, open the Registry and navigate to the key specified in Step 2. Ensure that the path supplied for *Library* is correct. If it is not, the *Epiphany Channels* object entry does not appear in the object list for the Performance Monitor.
4. From a command line, go to the *Win32* directory of the current instance and run:

```
unlodctr Epi  
lodctr EpiPerfmon.ini
```

where `unlodctr Epi` unloads parameters from any previous Registry entry.

EpiPerfMon.ini initializes the values associated with *Epi*.

Note: *Epi* is the default driver and key specified in **EpiPerfmon.reg**, which is described in *Monitoring Jobs* on page 53.)

You are now able to start the NT Performance Monitor and monitor the progress of *EpiChannel*. See *Monitoring Jobs* on page 53 for more information.

After Installing the Epiphany Software

After installing the Epiphany software you can:

- Run the Epiphany tools, such as EpiCenter Manager to configure your EpiCenter and to set up jobs for extracting source system data. Instructions are given in Chapter 3 of this *Guide*. Please read Chapters 1 and 2 for background information before attempting to use EpiCenter Manager.
- Enter your e-mail password to receive notification of Epiphany system status. You will use the Configuration dialog box in EpiCenter Manager to set this up. Instructions are given in Chapter 3.
- Use the Web Builder program to design ticksheets. Instructions are given in Chapter 4.
- Run the Epiphany extraction program (EpiChannel) to extract data from your source systems and place them in the Epiphany database (the EpiMart). EpiChannel performs the jobs you defined in EpiCenter Manager.

Epiphany Macros

This appendix describes the Epiphany-supplied system calls and SQL macros.

System Call Macros

For the most part, you will use Epiphany system call macros for the transfer of information related to the locations of files and database logins from the EpiMeta database to system calls. Many systems calls such as **mv** and **mkdir** are unable to read a database, and few system calls will be able to read Epiphany-created structures. Rather than leaving each system call to its own means to determine its appropriate information, EpiChannel may pass this information to the system call on the command line.

System Call Macro Syntax

The syntax for system-call macros can be either

\$\$VERB[*arg1, arg2, ...*]

or

\$\$VERB

Unless stated otherwise, the arguments are the names of roles defined with the job. You may use multiple arguments in most cases, which results in an

expansion of each argument. Note that the expansion of both of the following is the same:

```
$$verb[arg1, arg2]
$$verb[arg1] $$verb[arg2]
```

If the verb does not match one of the special words, no expansion of that verb occurs; for example:

```
echo $$notAverb
```

expands to

```
echo $$notAverb
```

If the argument is a role name, and the job does not define that role, the system call is considered to have an error.

Table B-1 describes the Epiphany system call macros.

Note: The Usage column in the following table stands for expected frequency of use.

Table B-1: Epiphany System Call Macros

Macro	Purpose	Usage	Definition
AGG	Child Procs	High	The name of the Aggbuilder executable in \$\$EPIBIN. For example: \$\$AGG \$\$EXEC_ARGS -j \$\$JOB_NAME
AGGVERIFY	Child Procs	Low	The name of the aggverify executable in \$\$EPIBIN.
APPSERVERHOST	Registry	Low	The value of this Registry variable.
APPSERVERPORT	Registry	Low	The value of this Registry variable.
CHARTSLOGFILE	Registry	Low	The value of this Registry variable.
CHARTSOUTPUTDIR	Registry	Low	The value of this Registry variable.

Table B-1: Epiphany System Call Macros

DATABASE	Database Login	High	Translates to the name of the database or instance. For example: isql /S \$\$SERVER[Tests] /U \$\$USER[Tests] /P \$\$PASSWORD[Tests] /d \$\$DATABASE[Tests] /w 300 /i /Q "gen_tests_run"
DBVENDOR	Database Login	Medium	Translates to the vendor of the database technology.
DEBUG_LEVEL	Command Line	Low	Translates to the current verbosity level of EpiChannel. Use this to pass EpiChannel's verbosity onto the subprocesses it spawns via system calls.
DIRNAME	File ID	Medium	Translates to the directory name of the data store, without the last filename component and without a trailing slash. If the role is <i>working dir</i> , the directory name's last component is a unique subdirectory generated for this particular run of EpiChannel. For example: echo DIRNAME is \$\$DIRNAME[Working Directory] (but with no arguments it is \$\$DIRNAME).
DSN	Database Login	Medium	Translates to the ODBC connection string for the database. This string may be generated even for databases accessed using native API's.
EPIBIN	Child Procs	Medium	The name of the <i>win32</i> directory under the <i>InstanceRootDir</i> Registry variable.
EXC	Child Procs	High	The name of the extract executable in \$\$EPIBIN.
EXC_ARGS	Child Procs	High	The recognized portions of the extract command line.

Table B-1: Epiphany System Call Macros

EXC_CMD	Child Procs	Medium	The extract program and its arguments other than job name. You can use this to fire sub- extract runs. For example: SSEX_CMD -j performance
EXCVERIFY	Child Procs	Low	The name of the excverify executable in S\$EPIBIN.
FILENAME	File ID	Medium	Translates to the filename of the data store. If the role is <i>Working Dir</i> , the file name is the name of the EpiChannel log file. For example: echo FILENAME is S\$FILENAME[Working Directory] (but with no arguments it is S\$FILENAME).
INSTANCE_NAME	Registry	Medium	The name of the instance's Registry subtree.
INSTANCEROOTDIR	Child Procs	Low	Value of the <i>InstanceRootDir</i> Registry variable.
ISS	Database Login	Low	Translates to a number associated with this source of information.
JOB_NAME	Child Procs	High	The name of the current job.
PASSWORD	Database Login	High	Translates to the password associated with the database login.
PATH	File ID	Medium	Translates to the directory name and file name placed together (in the DOS file path format).
PROGRAM_NAME	Child Procs	Low	The name of the current extract program.
REGISTRY_EPIPATH	Registry	Low	Name of the Epiphany Registry key.
REGISTRY_ROOT	Command Line	Low	Translates to the Registry key used by EpiChannel.
SERVER	Database Login	High	Translates to the database host server name (the machine's name) in the case of Microsoft SQLServer, or the SQLNet ID (sid) in the case of Oracle. SERVER and SQLNET are identical in behavior and can be interchanged.

Table B-1: Epiphany System Call Macros

SQLNET	Database Login	High	Translates to the database host server name (the machine's name) in the case of Microsoft SQLServer, or the SQLNet ID (sid) in the case of Oracle. SERVER and SQLNET are identical in behavior and can be interchanged.
USER	Database Login	High	Translates to the username associated with the database login.
VERSION	Database Login	Low	Translates to a release number or string associated with this database's installation.

Epiphany SQL Macros

Major database vendors have developed proprietary extensions to SQL that customers may choose to use in their SQL code because of their features. Epiphany also provides SQL macros for SQL Epiphany products. These macros, which are available as an option to customers, attempt to be database vendor-independent.

Epiphany supplies SQL macros to resolve these issues:

- Support coding of SQL statements that works with the syntax of multiple database engines.
- Support SQL that adapts to the structure of the star schema as defined via the EpiCenter Manager.
- Support extraction of contiguous but non-overlapping subsets from those tables that have dates or ascending identification fields.

Vendor-independent Macros

Although it is possible to avoid proprietary extensions to SQL, the features they offer may be necessary. As a result, consumers of SQL inherit problems raised by these differences unless they decide to bind themselves to a single database vendor. (For example, a function called NVL in one database is equivalent to a function called ISNULL in another database.)

Epiphany supports multiple database vendors, and Epiphany-executed SQL statements are a major consumer of SQL features. The Epiphany vendor-independent macros serve to provide some degree of isolation from database vendor differences. Epiphany uses these SQL macros in its own SQL so that it does not need to support different “codelines” of SQL.

Although these macros are available to use by Epiphany customers, their use is optional. Customers concerned only with a single database vendor might avoid all use of these macros. However, customers who use multiple vendors now or plan to in the future may choose to use the Epiphany SQL vendor-independence macros in the SQL statements they create.

SQL Macro Usage

An example of both preferred and less preferable SQL macro usage follows:

Preferred:

```
where COLUMN_FILTER[outfitting_order~,~outfitting_order_key~,~oo]
```

Less preferable:

```
where COLUMN_FILTER[ outfitting_order ~,~ outfitting_order_key ~,~ oo]
```

The following tables provide a brief explanation of each SQL macro:

- Table B-2 “Adaptive SQL Macros,” on page 231
- Table B-3 “Extraction Set Identification Macros,” on page 232
- Table B-4 “Smart Extraction Macros,” on page 235
- Table B-5 “Vendor-independent Macros,” on page 235
- Table B-6 “Testing Macros,” on page 242

SQL Macro Notes

- You can determine the “real” translations for most Epiphany macros by issuing the following SQL in an EpiMeta database:
Select * from translation_actual
- For usage examples of most of the SQL macros, see the initialization file *templates.sql*.
- The syntax for Epiphany SQL macros may be either `$$Macro[arg1~,~ arg2~,~ ...]` or `$$Macro`.
(Note that ~,~ is used to separate the arguments if there are multiple arguments in the list.)
- If an argument is a list of multiple elements, the elements are separated by commas. For an example, see the macro `CREATE_INDEX_IF_NOT EXISTS` in Table B-5.
- In some of the macro examples, column spaces have been added near the brackets and near the ~,~ entries for better formatting. In reality, any spaces that are present would be forwarded into the final SQL, so the best coding practice is to avoid them.
- The Usage column in the following tables stands for expected frequency of use.

Table B-2: Adaptive SQL Macros

Macro	Usage	Description
CURR	High	Expands the _A or _B suffix of the currently active tables. Allows you to reference the active or new EpiMart tables.
NEXT	High	Expands the _A or _B suffix of the <i>not</i> currently active tables.

Table B-3: Extraction Set Identification Macros

Macro	Usage	Description
COLUMN_FILTER [table_name ~,- column_name ~,- alias_name]	High	Expands to a "one-sided" SQL comparison expression that requires that alias_name.column_name be greater than or equal to the value in table table_name column column_name as of the start of the last run. This extracts "everything since last time." Although the alias name is optional: you should use it.
COLUMN_LAST_VALUE [tbl ~,- col]	High	Expands to the value of this column as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
COLUMN_RANGE_FILTER [table_name ~,- column_name ~,- alias_name]	High	Expands to a "one-sided" SQL comparison expression that requires alias_name.column_name to be greater than or equal to the value in table table_name column column_name as of the start of the last run, and less than or equal to this value as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running."
DATE_FILTER [column_name]	High	Expands to a "one-sided" SQL comparison expression that requires the column to be greater than or equal to the "current date/time" as of the start of the last run. This extracts "everything since last time." No table or alias arguments are available to the macro. If the column needs to be qualified, just add the qualifications in the first argument. For example: and \$\$DATE_FILTER[oo.date_key]

Table B-3: Extraction Set Identification Macros

DATE_RANGE_FILTER [column_name]	High	<p>Expands to a "two-sided" SQL comparison expression that requires the column to be greater than or equal to the "current date/time" as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running."</p> <p>This compares SQLServer datetimes. The column used for the comparison must be declared as a datetime or some variant in SQLServer. Only the date portion of the value is used: the time portion is discarded.</p>
DATE_LAST_VALUE	High	<p>Expands to the "current date" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.</p> <p>This compares SQLServer datetimes. The column used for the comparison must be declared as a datetime or some variant in SQLServer. Only the date portion of the value is used: the time portion is discarded.</p>
TIMESTAMP_FILTER [column_name]	High	<p>Expands to a "one-sided" SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run. This extracts "everything since last time."</p> <p>This compares SQLServer timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type in SQLServer, not as a time or date.</p>

Table B-3: Extraction Set Identification Macros

TIMESTAMP_FILTER_RANGE [column_name]	High	<p>Expands to a "two-sided" SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running."</p> <p>This compares SQLServer timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type in SQLServer, not as a time or date.</p>
TIMESTAMP_LAST_VALUE	High	Expands to the "current timestamp" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
YYYYMMDD_FILTER [column_name]	High	This is the same as a DATE_FILTER except that only the "day" portion of the date/times is used. (Some business semantics are most meaningful when applied to "days.") This extracts "everything since the last day, including the last day."
YYYYMMDD_FILTER_RANGE [column_name]	High	This is the same as a DATE_FILTER_RANGE except that only the "day" portion of the date/times is used. (Some business semantics are most meaningful when applied to "days.") This extracts "everything since the last day, including the last day, but not including today."
YYYYMMDD_LAST_VALUE	High	Expands to the "current date in YYYYMMDD format" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.

Table B-4: Smart Extraction Macros

Macro	Usage	Description
METADBNAME	Medium	Returns the name of the metadata database (EpiMeta).
MARTDBNAME	Medium	Returns the name of the datamart database (EpiMart).
INITIAL_LOAD	Medium	Expands to its argument if the initial load flag is set, or else expands to nothing.
NOT_INITIAL_LOAD	Medium	Expands to its argument if the initial load flag is set, or else expands to nothing.

Table B-5: Vendor-independent Macros

Macro	Usage	Description
ADD_MONTHS [date_expression - , ~ number]	Medium	Takes two arguments; adds the second as a number of months to the first argument, which is a date.
ASSERT_INDEX_EXISTS [index_name]	Low	Causes a SQL error if the index named in argument 1 does not exist. <pre> \$\$BEGIN_ASSERT_INDEX \$\$ASSERT_INDEX_EXISTS['XPKCustomerMap_B'] \$\$ASSERT_INDEX_EXISTS['XPKApplicationMap_B'] \$\$END_ASSERT_INDEX </pre>
BEGIN_ASSERT_INDEX	Low	In Oracle, declares the DECLARE INDEX_NOT_EXISTS exception. See ASSERT_INDEX_EXISTS.
CAT	High	Used as an operator to append "two" \$\$CAT 'strings'. <pre> \$\$TO_CHAR[table1.col1] \$\$CAT '-' \$\$CAT \$\$TO_CHAR[col2] </pre>
CHAR_1	Medium	Expands into a type definition for a single character field.

Table B-5: Vendor-independent Macros

CREATE_INDEX_IF_NOT_EXISTS [index_type~,~ index_name~,~ table_name~,~ column_list~,~ after_creation_clause]	Low	Creates an index if it is not already there. \$\$DDL_BEGIN \$\$CREATE_INDEX_IF_NOT_EXISTS[UNIQUE ~,~ XPK_123 ~,~ table1 ~,~ ss_key . iss . date_key . transtype_key . seq ~,~] \$\$DDL_END
DBNOW	High	Returns the date/time from the database engine. Select Col1 ss_key \$\$DBNOW date_modified from zork
DDL_BEGIN	Low	Starts a block of code that changes the schema. Use when there is no DECLARE statement already started. \$\$DDL_BEGIN \$\$NOT_DEBUG[\$\$DROP_TABLE_IF_EXISTS[table1]] \$\$DDL_END
DDL_BEGIN_NO_DECLARE	Low	Starts a block of code that changes the schema. Use when there is a DECLARE statement already started. DECLARE \$\$VAR[transFIXED] \$\$VARCHAR_50\$\$EOS \$\$DDL_BEGIN_NO_DECLARE \$\$VAR_ASSIGN_BEGIN[transFIXED] SELECT \$\$TO_CHAR[transtype_key] \$\$VAR_ASSIGN_INT0[transFIXED] FROM Transtype_O WHERE name = 'FINV_ADJUST' \$\$VAR_ASSIGN_END
DDL_END	Low	Ends a block of SQL that changes the schema. \$\$DDL_BEGIN \$\$DROP_TABLE_IF_EXISTS[\$\$FCTTBL[]\$\$NEXT] \$\$DROP_TABLE_IF_EXISTS[\$\$FCTTBL[]_INC] \$\$DDL_END

Table B-5: Vendor-independent Macros

DDL_EXEC [statement]	Low	<p>All items in the argument list are evaluated at runtime, not when the statement is parsed. This macro can construct SQL based on the values of variables computed in the same SQL block.</p> <pre> \$\$DDL_BEGIN \$\$DDL_EXEC[CREATE INDEX X_table1 ON table1 (iss, ss_key, date_key)] \$\$DDL_END </pre>
DECLARE_BEGIN	Low	<p>Starts a DECLARE block.</p> <pre> \$\$DECLARE_BEGIN \$\$DECLARE_BODY[\$\$VAR[count_INC] \$\$EPIINT] \$\$DECLARE_BODY[\$\$VAR[count_FC] \$\$EPIINT] BEGIN \$\$VAR_ASSIGN_BEGIN[count_INC] SELECT COUNT(1) \$\$VAR_ASSIGN_INTO[count_INC] FROM \$\$FCTTBL[]_INC \$\$VAR_ASSIGN_END </pre>
DECLARE_BODY [argument]	Low	Treats its argument as a declaration. See DECLARE_BEGIN .
DROP_INDEX [table_name~,~ index_name]	Medium	<p>Drops the index.</p> <pre> \$\$DDL_BEGIN \$\$DROP_INDEX[table1 ~,~ index_name] \$\$DDL_END </pre>
DROP_TABLE_IF_EXISTS [table_name]	Medium	<p>Drops the table without returning an error indicating that the table does not exist.</p> <pre> \$\$DDL_BEGIN \$\$DROP_TABLE_IF_EXISTS[table1] \$\$DROP_TABLE_IF_EXISTS[table2] \$\$DDL_END </pre>

Table B-5: Vendor-independent Macros

ELSE	Medium	The start of the negative clause of an IF statement.
END_ASSERT_INDEX	Low	Ends a block of checks that indexes exist. See ASSERT_INDEX.
END_IF	Medium	Ends an IF statement. see IF.
EOS	Low	Ends a SQL statement. SELECT 'PROCESSED'. COUNT(1), 1100 FROM table1 \$\$EOS
EPIINT	Medium	Declares an int. \$\$DECLARE_BEGIN \$\$DECLARE_BODY[\$\$VAR[unjoined] \$\$EPIINT] \$\$DECLARE_BODY[\$\$VAR[processed] \$\$EPIINT]
EPIKEY	Medium	Declares an Epiphany dimension key. See EPIINT.
FACTMONEY	Medium	Declares a monetary value. See EPIINT.
FACTQTY	Medium	Declares a decimal value. See EPIINT.
FLOAT	Medium	Declares a float value. See EPIINT.
IDENTITY	Medium	Declares a integer serial sequence. See EPIINT.
IF [condition]	Medium	Performs a conditional action. \$\$IF[\$\$VAR[fc_exists] = 0] \$\$DDL_EXEC[\$\$SELECT_INTO_BEGIN[temp_table] SELECT " \$\$SELECT_INTO_BODY[temp_table] FROM Old_table WHERE 1=0] \$\$END_IF \$\$DDL_END

Table B-5: Vendor-independent Macros

IJ_FROM [table_name1 ~,~ table_name2]	Low	Performs an inner join on two tables. See JOIN_WHERE.
JOIN_WHERE [join_condition]	Low	Supplies the WHERE clause for a join. SELECT col1, col2 FROM Table1 s \$\$LOJ_FROM[table2 m ~,~ s.iss = m.iss AND s.col2=m.col2] \$\$LOJ_FROM[table3 d ~,~ m.col1 = d.col1] WHERE 1=1 \$\$JOIN_WHERE[m.col1=d.col1(+)] \$\$JOIN_WHERE[s.iss = m.iss (+) AND s.col2 = m.col2 (+)]
LOJ_FROM [join_condition]	Low	Performs a left outer join. See JOIN_WHERE.
MAX_SYS_DATE	Medium	Returns the highest date supported by the database.
NO_FROM_LIST	Medium	Supplies the "dummy" FROM clause needed by some database vendors. SELECT 'MODIFIED', \$\$VAR[modified], 1050 \$\$NO_FROM_LIST\$\$EOS
NUMBER(9)	Medium	Declares a decimal(9). See EPIINT.
NUMBER(9,2)	Medium	Declares a decimal(9.2). See EPIINT.
NUMBER(9,5)	Medium	Declares a decimal(9.5). See EPIINT.
NVL [expression ~,~ value]	High	When the first argument is NULL, replace it with the value in the second argument. SELECT \$\$TO_CHAR[\$\$NVL[MAX(col1) ~,~ 1]]
ORACLE [expression]	High	Expands to nothing if the database is not Oracle. SELECT COUNT(1) FROM \$\$\$SQLSERVER[sysobjects]\$\$ORACLE[tabs]

Table B-5: Vendor-independent Macros

SELECT INTO_BEGIN [table_name]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. <pre> \$\$SELECT INTO_BEGIN[temp_tab] SELECT * \$\$SELECT INTO_BODY[temp_tab] FROM Old_tab WHERE 1=0 </pre>
SELECT INTO_BODY [table_name]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. See SELECT INTO_BEGIN.
SMALLDATE	Medium	Declares a SMALLDATETIME. See EPIINT.
SMALLINT	Medium	Declares a double-byte integer. See EPIINT.
SQLSERVER [expression]	High	Expands into nothing if the database engine is not SQLServer. <pre> SELECT COUNT(1) FROM \$\$\$SQLSERVER[sysobjects]\$\$ORACLE[tabs] </pre>
SSKEY	Low	Declares the type Epiphany uses for <i>ss_keys</i> . See EPIINT.
SUBSTRING [expression ~,~ start ~,~ length]	Medium	Performs a substring operation. For example: <pre> \$\$\$SUBSTRING[name ~,~1 ~,~8] </pre>
TABLE_EXISTS_ CONDITION [table_name]	Low	Detects if a table exists. <pre> SELECT COUNT(1) FROM \$\$\$SQLSERVER[sysobjects]\$\$ORACLE[tabs] WHERE \$\$TABLE_EXISTS_CONDITION[table_name] </pre>

Table B-5: Vendor-independent Macros

TABLE_WITH_PREFIX [database_name ~,~ table_name]	Medium	Qualifies a table name with a database or user name. SELECT source_system_key iss FROM \$\$TABLE_WITH_PREFIX[\$\$METADBNAM ~,~ source_system]
TINYINT	Medium	Declares a single-byte integer. See EPIINT.
TO_CHAR [expression]	High	Converts a value to a character. Length is second argument. SELECT \$\$TO_CHAR[\$\$NVL[MAX(col1) ~,~ 1]]
TO_DATE [expression]	medium	Converts a value to a database date.
TO_EPIDATE [expression]	High	Converts a date to the string format preferred by EpiChannel. Select col1 ss_key. \$\$TO_EPIDATE[date_col] date_modified from zork
TO_YYYYMMDD [expression]	Medium	Converts a data to a YYYYMMDD string.
VAR [variable_name]	Medium	References a database variable. SELECT 'PROCESSED', \$\$VAR[processed], 1100 \$\$NO_FROM_LIST\$\$EOS
VAR_ASSIGN_BEGIN [variable_name]	Medium	Assigns to a database variable. \$\$VAR_ASSIGN_BEGIN[max_key] SELECT \$\$TO_CHAR[\$\$NVL[MAX(col1) ~,~ 1]] \$\$VAR_ASSIGN_INT0[max_key] FROM table2 \$\$VAR_ASSIGN_END.
VAR_ASSIGN_END	Medium	Assigns to a database variable. See VAR_ASSIGN_BEGIN.
VAR_ASSIGN_INT0 [variable_name]	Medium	Assigns to a database variable. See VAR_ASSIGN_BEGIN.

Table B-5: Vendor-independent Macros

VARCHAR_100	Medium	Declares a variable-width character datatype that holds a maximum of 100 characters. See EPIINT.
VARCHAR_15	Medium	Declares a variable-width character datatype that holds a maximum of 15 characters. See EPIINT.
VARCHAR_25	Medium	Declares a variable-width character datatype that holds a maximum of 25 characters. See EPIINT.
VARCHAR_255	Medium	Declares a variable-width character datatype that holds a maximum of 255 characters. See EPIINT.
VARCHAR_5	Medium	Declares a variable-width character datatype that holds a maximum of 5 characters. See EPIINT.
VARCHAR_50	Medium	Declares a variable-width character datatype that holds a maximum of 50 characters. See EPIINT.

Table B-6: Testing Macros

Macro	Usage	Description
DEBUG	Low	Expands to nothing if the extract command's verbosity level is not less than 3, otherwise returns its argument.
NOT_DEBUG	Low	Expands to nothing if the extract command's verbosity level is not less than 3, otherwise returns its argument

EpiCenter Configuration

The Configuration data that appears in the EpiCenter Manager's Configuration dialog box has been set for a default EpiCenter. Other than entering your e-mail password, the information should be correct, or require minimal alteration.

To modify Configuration settings:

Choose Configuration from the EpiCenter menu. The Configuration dialog box (see Figure C-1) is displayed. It has three tabs: Configuration, Transaction Type, and Measure Units. *[The Measure Units tab is not supported for this release.]*

Configuration

A description of the Configuration fields is given in the Configuration dialog box. You may modify these settings if necessary. Select the key in the list and enter a new value in the Value textbox, and click Update.

Note the following:

- You need to change the mail password after installing the Epiphany software.

EpiCenter Configuration

- `current_datamart A / B`

Indicates which set of tables (A or B) is active. See *Mirroring: A and B Tables* on page 55

- `date_type`

Calendar is the default.

`date_445` represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.

- `fiscal_year start_m[onth]`

If the chosen month is other than January, then the actual starting year is one less than the `start_year` value (thus the given year may start on January 1 and still be a complete fiscal year).

- `number_of_years`

The number of years during which the data warehouse will be operational. The default is 20.

- `start_day_445`

The beginning day of the 445 quarter.

- `start_year`

The year that the EpiMart becomes operational. The default is 1990.

- `version`

The version number of this EpiCenter Manager.

- `week_start_day`

The day of the week that is the first day of the week. Sunday is the default.

09625518 072500

- last_extract_date
The date that appears as the *Data is valid as of...* in Clarity reports.

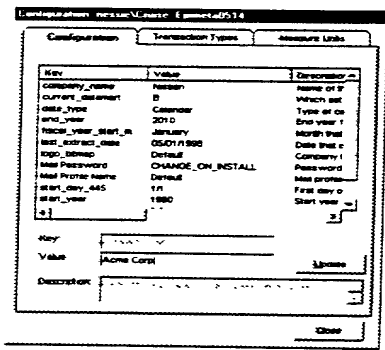


Figure C-1 Configuration Tab

Transaction Types

The current transaction types are shown in the Transaction Types tab (Figure C-2) of your Configuration dialog box. These are the typical transaction types for a generic installation. Your site may need additional transaction types for complicated measures.

Use this tab to customize the transaction types for your site. Keys 1-99 are reserved for booking transaction types. Keys 101-199 are reserved for shipping transaction types.

After modifying the transaction types, click Update, which writes the new data to the EpiMeta.

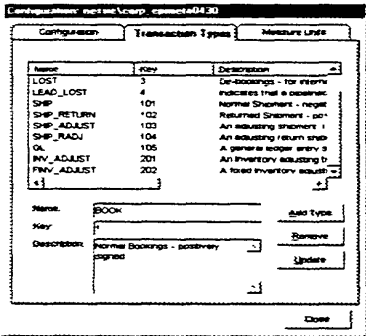


Figure C-2 Transaction Type

Measure Units

[The Measure Units tab is not supported for this release.]

Measure units define how currency units, percentages, and numerical units are displayed by default on ticksheets. Enter the name of the unit and a description in the textboxes in the Measure Units tab and click Add Unit. The system generates a hidden key for the unit. Web Builder uses this key when flagging a measure with a unit designation.

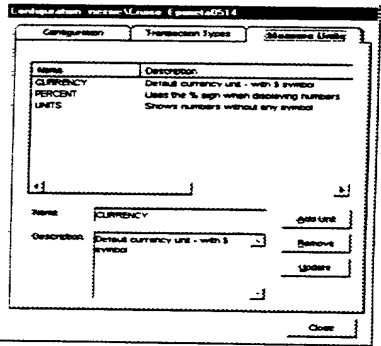


Figure C-3 Measure Units Tab

EpiCenter Configuration

09:25:40.072500

APPENDIX D

Date Dimension Fields

The date dimension fields used by the Epiphany system are described below.

Table D-1: Date Dimension Fields

dim_col_name	Description
cq_and_cy_name	Calendar quarter and year name. For example, Q1 1998.
cq_name	Calendar quarter name. For example, Q1.
cy_name	Calendar year name. For example, 1998.
date_key	Primary key—date as a native date type.
day_cq_begin	Whether or not this is a day on which a calendar quarter begins. (1/0).
day_cq_end	Whether or not this is a day on which a calendar quarter ends. (1/0).
day_cy_begin	Whether or not this is a day on which a calendar year begins. (1/0).
day_cy_end	Whether or not this is a day on which a calendar year ends. (1/0).

Table D-1: Date Dimension Fields

dim_col_name	Description
day_fq_begin	Whether or not this is a day on which a fiscal quarter begins. (1/0).
day_fq_end	Whether or not this is a day on which a fiscal quarter ends. (1/0).
day_fy_begin	Whether or not this is a day on which a fiscal year begins. (1/0).
day_fy_end	Whether or not this is a day on which a fiscal year ends. (1/0).
day_month_begin	Whether or not this is a day on which a month or period begins. (1/0).
day_month_end	Whether or not this is a day on which a month or period ends. (1/0).
day_name	The day as a native date type.
day_name_char	Date (as a string). For example, Apr 02 1995.
day_name_char_weekday	Date (as a string) with weekday prefix. For example, Sun Apr 02 1995.
day_number_in_cq	The number of this day in the calendar quarter, starting at 1.
day_number_in_cy	The number of this day in the calendar year, starting at 1.
day_number_in_fq	The number of this day in the fiscal quarter, starting at 1.
day_number_in_fy	The number of this day in the fiscal year, starting at 1.

Table D-1: Date Dimension Fields

dim_col_name	Description
day_number_in_month	The number of this day in the month or period, starting at 1.
day_number_in_week	The number of this day in the week, starting at 1.
day_number_til_end_cq	The number of days until the end of the calendar quarter, ending with 1.
day_number_til_end_fq	The number of days until the end of the fiscal quarter, ending with 1.
fq_and_fy_name	Fiscal quarter and year name. For example, Q1 1998.
fq_name	Fiscal quarter name. For example, Q1.
fy_name	Fiscal year name. For example, 1998.
month_and_cy_name	Month (abbrev.) name and calendar year. For example, Apr 1998.
month_and_fy_name	Month (abbrev.) or period name and fiscal year. For example, Apr 1998.
month_number	Month or period number since the first date in the system, starting at 1.
month_number_in_cq	Month number since the start of the calendar quarter, starting at 1.
month_number_in_cy	Month number since the start of the calendar year, starting at 1.
month_number_in_fq	Month or period number since the start of the fiscal quarter, starting at 1.

Table D-1: Date Dimension Fields

dim_col_name	Description
month_number_in_fy	Month or period number since the start of the fiscal year, starting at 1.
month_number_til_end_cy	Number of months or periods until the end of the calendar year, ending with 1.
month_number_til_end_fy	Number of months or periods until the end of the fiscal year, ending with 1.
vacation_day	Whether or not this day is a vacation, (1/0).
week_friday	Date (as a string) for the Friday of the current week. For example, Apr 01 1994.
week_monday	Date (as a string) for the Monday of the current week. For example, Apr 01 1996.
week_number	Week number since the first date in the system, starting at 1.
week_number_cq	Week number since the start of the calendar quarter, starting at 1.
week_number_cy	Week number since the start of the calendar year, starting at 1.
week_number_fq	Week number since the start of the fiscal quarter, starting at 1.
week_number_fy	Week number since the start of the fiscal year, starting at 1.
week_number_til_end_cq	Week number until the end of the calendar quarter, ending with 1.
week_number_til_end_cy	Number of weeks until the end of the calendar year, ending with 1.

09625518.072500

Table D-1: Date Dimension Fields

dim_col_name	Description
week_number_til_end_fq	Week number until the end of the fiscal quarter, ending with 1.
week_number_til_end_fy	Number of weeks until the end of the fiscal year, ending with 1.
week_saturday	Date (as a string) for the Saturday of the current week. For example, Apr 01 1995.
week_sunday	Date (as a string) for the Sunday of the current week. For example, Apr 02 1995.
weekday	Weekday prefix. For example, Sun.

Date Dimension Fields

09/25/18 07:25:00

APPENDIX E

Physical Type Values

Table E-1, *Oracle Physical Type Values*, and Table E-2, *SQLServer Physical Type Values*, define the database type translations for the physical types you select in EpiCenter Manager's Base Dimension and External Tables dialog boxes. The physical type you select is replaced by its associated database type.

The translation of physical type to database type depends on your RDBMS platform.

Table E-1: Oracle Physical Type Values

Physical Type	Database Type
CHAR_1	CHAR(1)
EPIINT	NUMBER(9)
EPIKEY	NUMBER(9)
FACTMONEY	NUMBER(16,4)
FACTQTY	NUMBER(16,4)
FLOAT	FLOAT
IDENTITY	NUMBER(9)
NUMBER(9)	NUMBER(9)

Table E-1: Oracle Physical Type Values

Physical Type	Database Type
NUMBER(9,2)	NUMBER(9,2)
NUMBER(9,5)	NUMBER(9,5)
SMALLDATE	DATE
SMALLINT	SMALLINT
SSKEY	VARCHAR2(50)
TINYINT	SMALLINT
VARCHAR_5	VARCHAR2(5)
VARCHAR_15	VARCHAR2(15)
VARCHAR_25	VARCHAR2(25)
VARCHAR_50	VARCHAR2(50)
VARCHAR_100	VARCHAR2(100)
VARCHAR_255	VARCHAR2(255)

Table E-2: SQLServer Physical Type Values

Physical Type	Database Type
CHAR_1	CHAR(1)
EPIINT	INT
EPIKEY	INT
FACTMONEY	MONEY
FACTQTY	DECIMAL(16,4)
FLOAT	FLOAT
IDENTITY	INT IDENTITY
NUMBER(9)	DECIMAL(9)
NUMBER(9,2)	DECIMAL(9,2)
NUMBER(9,5)	DECIMAL(9,5)
SMALLDATE	SMALLDATETIME
SMALLINT	TINYINT
SSKEY	VARCHAR(50)
TINYINT	SMALLINT
VARCHAR_5	VARCHAR(5)
VARCHAR_15	VARCHAR(15)
VARCHAR_25	VARCHAR(25)
VARCHAR_50	VARCHAR(50)
VARCHAR_100	VARCHAR(100)
VARCHAR_255	VARCHAR(255)

Physical Type Values

062518 072500

Writing Staging SQL Statements

The Epiphany extraction process (excluding aggregation) consists of two phases: 1) loading data from source systems into the Epiphany staging tables, and 2) running semantic instances against these staging tables. This appendix describes the recommended practices for writing SQL statements that populate the dimension and fact staging tables.

Base Dimension Staging SQL Statements

Base dimensions describe the physical dimension tables in EpiMart. You can use EpiCenter Manager to configure the dimension columns for each base dimension table. During an extraction job, one or more SQL statements can be executed against the base dimension staging table to populate these columns.

The purpose of the base dimension staging query is to extract the latest values from the source system. You need not be concerned with determining when a value has changed because Epiphany's semantic templates perform this task.

You can use the Template feature in the SQL Statement dialog box (see Figure 3-19, page 98) to provide an SQL template for a given dimension table. For example, assume you define a base dimension called Customer with the following dimension columns:

- FullName
- Age
- City
- State
- ZipCode

In the Tables References panel of the SQL Statement dialog box, select the option *Populates a dimension*. Choose the dimension table (Customer) from the drop-down list. Clicking the Template button will display the following SQL in the dialog box:

```
SELECT
    <YOUR EXPRESSION> customer_skey,
    <YOUR EXPRESSION> data_modified,
    <YOUR EXPRESSION> FullName,
    <YOUR EXPRESSION> Age,
    <YOUR EXPRESSION> City,
    <YOUR EXPRESSION> State,
    <YOUR EXPRESSION> ZipCode
FROM
    <YOUR TABLE>
```

This is a regular SQL statement for which you must provide the values <YOUR EXPRESSION> and <YOUR TABLE>. You must assign each column in the SELECT list a valid SQL expression for the value of its destination column. A FROM clause is required to make this a valid statement; a WHERE clause is optional.

The first two columns, *customer_sskey* and *date_modified*, were not specified in the definition of the Customer base dimension. These columns are implicitly added to the base dimension table by Epiphany's Adaptive Schema Generator and must be populated by the staging SQL. The first column is a source system key (*sskey*) and should be unique for every row in the staging table. The concept of *sskey* is important in EpiMart because it tells the semantic templates whenever a row in the staging table represents the same source system entity as a row that already exists in the dimension table. The *sskey* is a variable length string, normally of maximum length 50. It corresponds to the primary key of the source system table or the tables that make up this query.

Note: If the *sskey* column is of interest to end users for querying, then a dimension column populated with the same values will need to be created because the *sskey* is not available for ticksheet configuration.

The other implicit column, *date_modified*, has the same name in all base dimension staging tables and is used to identify when a base dimension row is inserted into the EpiMart. If the source system contains a *creation date* field, then this field should be used. Otherwise, you can use the source system's expression for "right now," which causes newly extracted rows to assume the date when they were extracted into EpiMart; for Microsoft SQLServer this expression is `GetDate()`. For best results, dates should be returned as strings, for example, 5/1/1998.

The remainder of the columns in the SELECT list must be populated with an appropriate expression for the meaning of that column. Any SQL expression that can be executed against the source RDBMS is valid. Also, Epiphany provides a set of SQL macros that will be automatically expanded to the correct syntax for your source system. Use of macros facilitates the cross platform usage of your SQL Statements. See Appendix B, *Epiphany Macros*, for more information.

An important point about these expressions is that null values are *not* allowed in any field of the staging table. The reason for this is simple: the Epiphany system uses GROUP BY statements at end-user query time to form the tables and charts of front-end applications such as Clarity. However, fact rows that aggregate on columns with null values are left out of the resulting reports because nulls are removed from GROUP BY's. Rather than incurring the query-

time penalty for this check, EpiMart insists on non-null dimension column values.

To circumvent this problem, substitute the string UNKNOWN for any null values using the NVL macro. The Epiphany system will automatically generate an UNKNOWN row in your dimension table. The UNKNOWN value is configurable; if UNKNOWN is a valid value in your dimension table, use another value.

Duplicate sskey's

If during a single extraction, a staging table is loaded with two or more rows with the same *sskey*, then the last row entered is used. See Appendix G for a description of the dimension semantic types.

Dimension Staging Queries with Joins

The Epiphany system allows the use of joins in base dimension staging queries. Star schemas typically de-normalize data structures in transactional systems into flat hierarchies, and you must be aware of what the granularity of a base dimension represents in this circumstance.

For example, you will rarely want to use a Cartesian product of two tables in a base dimension staging query, unless the *sskey* of the result set will combine the primary keys of the two tables that are being crossed. It is more common for a single table to “drive” the result set, with other tables joined through unique key lookups to provide additional textual values. For instance, a Product Master table in the source system might represent the driving table of a Product base dimension (with the *sskey* taken from the primary key of the Product Master table), but other tables with textual values for Product Line or Platform may be joined with this master table. In this case, you should ensure that the joined columns of the lookup tables are properly indexed (usually with UNIQUE indexes).

Constructing Base Dimension Queries with DISTINCT Fact Values

Sometimes dimensions are created for which no corresponding master table exists in the source system. For instance, an Order fact may have an Order type associated with it (with several possible choices). These values will be embedded directly in the fact rows on the source, but no lookup table exists with all the choices. In this case, a SELECT DISTINCT query against the source system's fact table might be appropriate for populating base dimension staging tables in EpiMart. The alternative to this method is the use of degenerate dimensions in the fact table, although degenerate dimensions cannot be aggregated.

Fact Staging SQL Statements

SQL statements that populate fact staging tables are generally more complex than the ones used to load dimension staging tables. As with base dimension tables, the columns of the SELECT statements are determined by the metadata definition of the fact table (and its constellation) along with certain implicit rules.

To illustrate this point, assume that you define an Order fact in a Sales constellation with the following dimension roles:

- CustomerBillTo
- Product
- CustomerShipTo
- SalesPerson

The constellation contains a single degenerate dimension called OrderNumber, and the Order table has two fact columns: *net_price* and *number_units* that represent the extended amount for an order line item, along with the quantity.

Clicking the Template button on the SQL Statement dialog box (with the *Populates fact table* option selected and the Order table selected in the drop-down list) displays the following SQL in the dialog box:

```
SELECT
  <YOUR EXPRESSION> ss_key,
  <YOUR EXPRESSION> date_key,
  <YOUR EXPRESSION> transtype_key,
  <YOUR EXPRESSION> process_key,
  <YOUR EXPRESSION> customerbillto_sskey,
  <YOUR EXPRESSION> product_sskey,
  <YOUR EXPRESSION> customershipto_sskey,
  <YOUR EXPRESSION> salesperson_sskey,
  <YOUR EXPRESSION> ordernumber_key,
  <YOUR EXPRESSION> net_price,
  <YOUR EXPRESSION> number_units
FROM
  <YOUR TABLE>
```

As with base dimension staging queries, you must identify what a row in this fact table represents. Based on the columns in this example, a row seems to indicate a line item of a sales order. (In this case, the assumption is that the salesperson gets full credit for a line item; another interpretation of this fact row might be a particular amount of credit that a salesperson received for an order line item.) Typically, the FROM clause of this query would join the Order Line Item table to the Order Header table in the source system.

The columns in the SELECT list can now be divided into these categories:

- Implicit columns that were added automatically
- Dimension role foreign keys
- Degenerate dimension keys
- Fact numeric columns

First, consider implicit columns. As with base dimensions, each fact staging row contains an *ss_key* (notice the difference in spelling) that uniquely identifies this row in the source system. In this example, the *ss_key* might be a concatenation of the Order Number with the Order Line Number (since this combination is presumably unique). *ss_key*'s will be used on subsequent extractions to prevent duplicate copies of the fact row from being created in EpiMart.

The *date_key* indicates when the fact occurred. Since time is a central component of EpiMart, each fact table must contain this column. Many facts are time based; in this example, *date_key* represents the time when the order was placed. However, if time is not important for this fact, then the current system time can be used as a placeholder. Note that *date_key* is granular only to that single *day* when the fact occurred. For best results, the fact SQL Statement should return the day as a string, for instance, 5/1/1998.

Transaction type is another central concept of fact table processing in EpiMart. The SQL statement should return a numeric key that matches with one of the transaction types defined on the Configuration dialog box in EpiCenter Manager. (See Appendix G, *Semantic Types*, for more information about *transtype_key*.)

The *process_key* identifies rows from the fact table to be processed by a specific semantic type. (A fact table can contain different types of rows, requiring different semantic types.) For example, the Order fact might hold both Bookings and Shippings, and *process_key* would identify which fact staging rows were which. (See Appendix G, *Semantic Types*, for more information about *process_key*.)

Next, you must enter values for each of the dimension role foreign keys. Notice that the names of the columns in the SQL template are *DimRoleName_sskey*. These fields refer back to *sskey*'s of the base dimension tables for this fact. You need to understand the meaning of each base dimension table to ensure that the keys resolve properly. If the *sskey* of the Product base dimension is taken from a Product Master list in the source system, then *product_sskey* in the Order table must also refer to an entry in the Product Master. If a base dimension is the cross-product of two source system tables, the fact staging keys for that dimension must also represent a unique cross-product entry.

Degenerate keys in the fact staging query should be populated with string values. In the example above, the *ordernumber_key* field would probably be populated with the actual primary key of the Order Header table, such as Order Number 253AD56.

Finally, the numeric columns represent the actual quantities and raw amounts that are associated with each fact entry. Each column should be an additive amount for correct front-end query results. For instance, total dollar amounts for a line item should be populated instead of unit prices because unit prices cannot be added across fact rows.

Using External Tables as Inputs to Staging Queries

Sometimes it may be necessary to bring data into EpiMart external (temporary) tables before performing any joins; for example, if the source system's SQL limits your ability to manipulate the data. The full power of EpiMart's RDBMS engine can then be used to load the staging tables. In this case, the following sequence of actions is usually employed:

1. Drop any indexes on the external tables for fast loading.
2. Load the external tables from the source system.
3. Create any indexes on external tables needed for fast joins.
4. Load the staging tables using queries against the EpiMart External tables. All query plans should use the indexes built in Step 3.

Semantic Types

This appendix describes the dimension and fact semantic types.

Note: Fact rows with all zero facts are discarded by all semantics.

Dimension Semantic Types

In the initial release of Clarity and Relevance 3.2, there is the only one dimension semantic type: Slowly Changing Dimensions.

The Release 3.2 Service Pack 1 adds these dimension semantic types: Latest Dimension Value, First Dimension Value, and Initial Dimension Value.

Slowly Changing Dimensions

A Slowly Changing Dimension is a dimension in which the attributes or hierarchy of the dimension can change over time, but historical data is not restated. Two examples follow:

- A sporting goods chain decides to rename a store. By using the Slowly Changing Dimension semantic type, the old name is retained for all of the historical data. One can still identify when the store changed names and compare sales before and after the name change.

Semantic Types

- This same chain is national and has stores divided into three regions. On January 1, 1998, the chain reorganizes its regions, moving Denver from the Central to the Western region. Their 1998 sales forecasts take into account that the Denver store is in the Western region, but they do not want to recalculate forecasts and actuals from previous years. Using the Slowly Changing Dimension, sales for Denver can be aggregated up to the Central region through 1997. Beginning January 1, 1998, Denver sales are applied to the Western Region.

The Slowly Changing Dimension semantic type accomplishes the following logic:

- Rows with the same *sskey* in the staging table will be eliminated following in a “last in wins” rule. This is determined by the special column called *ikey*, which is created by EpiChannel and is automatically incremented during normal extractions. The highest *ikey* row for a given *sskey* will be accepted.
- New rows are created by searching through the dimension staging table for new *sskey* values, or *sskey* values that have one or more dimension column changes from the last known values. Each of these cases creates a new row in the dimension table. The mapping row for that *sskey* points to the latest dimension row with that *sskey* value.
- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of changed rows is reported in the *Modified* column. The number of rows in the staging table is reported in the *Processed* column.
- The dimension table has its key column made into a Primary Key index. Each dimension column is also Indexed (non-uniquely). The Mapping table is indexed (primary key) on *iss*, *sskey*.

iss is used when two source systems have the same *sskey* values (such as when the SAP and Vantive systems both have a Customer #2 record). *iss* is determined by the source system identifier selected using the General tab of the Data Store dialog box (Figure 3-15, page 91).

- The column *dimension_key_REAL* (where *dimension* is the dimension column name) is set to the first key value for each *sskey*. In other words, when a new *sskey* is discovered, then the *REAL* key is set to the new dimension key value. Subsequent dimension rows for this *sskey* will retain the original *REAL* key value.
- The UNKNOWN *sskey* always maps to dimension key value 1 in the Mapping table.

You need to be aware of the following:

- Do not allow dimension column values to *oscillate* unpredictably. (See *First Dimension Value* on page 270 for more information.) In particular, do not rely on *ikey* filtering of duplicate *sskey* values if two or more rows during a single extraction might have different values for one or more dimension columns. The reason is that a new row will be created in the dimension table for every extraction for which a change is recorded. This can cause two values to “compete” with each other, forcing an unending sequence of row creation in the dimension table.
- The only way to remove rows from dimension tables once they have been extracted is with an explicit delete or truncation.

Latest Dimension Value

(Release 3.2 Service Pack 1)

The Latest Dimension Value semantic type applies changes retroactively to a dimension. Thus the changes take effect for all historical data, as well as for current and future loads.

For example, assume that a sporting goods store has a category historically called *Rollerblades*. Now that they are selling other brands, the store wants to change the category to *In-line Skates*. By using the Latest Dimension Value semantic type, this change can affect all of the historical data because all previous sales of *Rollerblades* are now labeled *In-line Skates*. As a result, the store can compare year-to-year sales of all in-line skates.

Semantic Types

This semantic type has the same duplicate *sskey* filtering as Slowly Changing Dimensions. Use this semantic type for an implementation that “restates history” when a source dimension table changes.

Note the following:

- New *sskey*'s are inserted in both the dimension table and mapping table. Existing *sskey*'s with one or more changed dimension columns are updated in place in the dimension table (that is, the same dimension row is used) with the latest values.
- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of changed rows is reported in the *Modified* column. The number of rows in the staging table is reported in the *Processed* column.
- Latest Dimension Value has the same indexing as Slowly Changing Dimensions.
- The REAL column is always equal to the dimension key.
- Latest Dimension Value has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.

First Dimension Value

(Release 3.2 Service Pack 1)

The First Dimension Value semantic type ignores any changes to a dimension. The values that were present when the row was first inserted are preserved forever, regardless of any future changes.

You might want to use this type if your source data comes from two systems that are not in complete agreement with each other. For instance, if one system has Customer #12345 as *David Anderson*, and the other has the same customer as *David Andersen*. Ideally, you would determine which one was in error and correct it.

In the meantime, you could choose to apply the first value read and to ignore the other. (This is a good method for avoiding the *oscillation* problem mentioned on page 269.) If you were to use the Slowly Changing Dimension semantic type in this case, there would be a race between the two source systems for each extraction, and (in the worst case), your dimension values could alternate between the two values with every extraction.

Note the following:

- First Dimension Value has the same duplicate *sskey* filtering as Slowly Changing Dimensions.
- New *sskey*'s are inserted in both the dimension table and the mapping table. Existing *sskey*'s are ignored.
- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of rows in the staging table is reported in the *Processed* column.
- First Dimension Value has the same indexing as Slowly Changing Dimensions.
- The REAL column is always equal to the dimension key.
- First Dimension Value has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.

Initial Dimension Load

(Release 3.2 Service Pack 1)

The initial Dimension Load semantic type is used to load the dimension without regard to any previously existing rows. This can be used for the initial load of the empty EpiMart, and also to completely reload a dimension, ignoring existing values. Using any other semantic type would require emptying the existing dimension table before beginning the extraction.

Semantic Types

Note the following:

- Initial Dimension Load has the same duplicate *sskey* filtering as Slowly Changing Dimensions.
- The existing dimension and mapping tables are ignored; all *sskey*'s are imported directly into the dimension and the mapping tables.
- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of rows in the staging table is reported in the *Processed* column.
- Initial Dimension Load has the same indexing as Slowly Changing Dimensions.
- The REAL column is always equal to the dimension key.
- Initial Dimension Load has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.

Fact Semantic Types

Update Unjoined (Optional)

Always run this semantic type *before* any other semantic type for a fact table. Its purpose is to check the referential integrity of all dimension *sskey*'s in the fact staging table and to set all those keys that do not resolve to valid dimension *sskey*'s to the special value UNKNOWN.

The special *sskey* value of UNKNOWN always maps to the dimension key value 1 (the UNKNOWN dimension row). By setting all unresolved dimension *sskey*'s to UNKNOWN, those fact entries will be mapped to this special dimension row. All reports run against these fact rows will yield the default values for that dimension (see *The UNKNOWN Dimension Row* on page 34 and *The Update Unjoined Semantic Type* on page 36 for more information).

Update Unjoined uses the dimension mapping tables to determine whether a given dimension *sskey* is valid. For this reason, all dimension semantic instances must be executed before any fact semantic type.

In the EpiChannel log file, the number of changed rows is reported in the *Modified* column.

Custom Fact Index

In EpiCenter Manager, you use the Custom Index tab of the Fact Table dialog box to define custom indexes for a fact table. The semantic type Custom Fact Index, however, must be executed for the fact table as part of the scheduled extraction process in order for these indexes to be built.

Important: Schedule this semantic Instance *after* all other semantic instances for a fact table.

For a discussion of custom indexing, see *Custom Fact Indexing* on page 33.

Transactional

The Transactional Fact Semantic Type is the simplest means of moving fact staging data into fact base tables. This semantic type uses the following logic:

- Only rows with *process_key* set to 1 (Transactional) are used; others are discarded.
- For *sskey*'s that are already entered in the current fact base table, all rows in the staging table with that *sskey* and with the same or earlier *date_key* are discarded. If the *sskey* already exists in the fact table, only later dates can be added to the fact table.

Two or more rows with the same *sskey* can be imported into the fact base table if they arrive in the same extraction and the *sskey* either does not already exist, or exists but is dated earlier. (This property is used by the *pseudo-order* approach to the Booking/Shipping problem; see *Transactional/State-like/Force Close* on page 275.)

Semantic Types

- In the EpiChannel log file, all rows added to the fact base table are reported in the *Inserted* column, and the total number of rows in the staging table is reported in the *Processed* column. Any rows whose dimension *sskey*'s do not resolve to valid values are reported as Unjoined. (This occurs only if Update Unjoined is not used; see *Update Unjoined (Optional)* on page 272.)

Note the following:

- Transactional semantics should be used for event facts; once the fact event has occurred, it can never be modified.
- To reload transactions after they have been loaded into the fact base table, the rows must be deleted from the fact base table, or the fact base table must be truncated.

Transactional/State-like

The Transactional/State-like semantic type allows changes to already existing rows in EpiMart. It uses the same logic as the Transactional semantic type, with the addition of the logic described below.

First these three steps occur:

1. Records in the staging table with *process_key* of 2 (state-like) are treated as Orders that can be *differenced* between extractions (a discussion of this follows).
2. For records in which *process_key* = 2, duplicate *sskey*'s with the same *date_key* in the same extraction cause only the highest *key* value to be used. Other rows are discarded. This is the same filtering that happens in dimension semantics such as Slowly Changing Dimensions.

3. For an *sskey* with the highest *ikey* (which means that for every set of rows with the same *sskey*, take the row with the highest *ikey*), if the *date_key* for that staging row is less than the last *date_key* for that *sskey* in the fact base table, the staging row is discarded. In other words, an Order can only be modified on its last reported date, or some time further into the future.

After Steps 1-3, the staging fact columns and dimension values are compared with the current values in the fact table (if any). Adjusting records are created in the fact table so that the fact table now reflects the reported *state* from the staging table. (This is why the term *state-like* is used.) *Differenced* transactions are invented if the numeric fact columns have changed.

If the dimensionality has changed, then the Order will be “de-booked” and “re-booked” with the correct dimensionality.

If the same *sskey* appears in the staging table with more than one *date_key*, then further adjusting transactions are made in the fact base table as appropriate to bring the fact base table in line with the reported staging rows.

Note: By convention, Bookings are entered with positive facts (negative for Returned Orders), whereas Shipments are entered with negative facts (positive for Returned shipments).

When using this semantic type it is difficult to ensure that Backlog calculations will remain consistent when Orders are not completely closed in the source system. Use the Transactional/State-like/Force Close instead.

Transactional/State-like/Force Close

This semantic type is equivalent to Transactional/State-like with the following additional logic.

Once a Booked Order is entered into EpiMart, it remains in that state (with an Open Backlog) forever. In normal scenarios, an invoice eventually arrives, which will close the Backlog. However, in some source systems, Booked Orders can be removed from the system completely. If such an Order had been entered previously into EpiMart, then it will remain in an Open Backlog condition forever.

The solution to this problem is to use Transactional/State-like/Force Close, which establishes a "Challenge Protocol" for Open Orders. In this scenario, all Open Bookings must be extracted into the staging table during every extraction. The reason is that the Force Close logic will close out all Open Orders (*sskey* 's with non-zero facts in the system) that do not appear in the fact staging table. Only Booking Transaction types (those with *transtype_key* values between 1 and 99) will be affected in this manner.

When using this semantic type, the methodology that has been found to work in practice involves the use of pseudo-orders as follows:

- One extraction SQL statement extracts all Open Orders. Fact amounts are the Open amounts (what has not been shipped yet), not the ordered amounts. Transaction types for this statement should be in the Booking range (1...99), and *process_key* should be 2 (state-like).
- The second extraction statement uses *process_key* of 1 (Transactional) and represents all shipments in the system. These records normally go into the system with negative facts for positive shipments (by convention). These transaction types should be in the shipment range (101 or greater).
- The third statement is a restatement of the shipment, but as a Booking (*transtype_key* between 1...99). The *process_key* is still 1 (allowing the Transactional Semantics to import it), but the transaction type is a Booking. The same *sskey* and dimensionality as the second statement are used. These are the pseudo-orders since they are actual shipments that are entered as Orders.

The net effect of this methodology is that as a shipment is reported against Bookings, the Open Booking quantity in statement 1 is decremented, while the actual shipment is restated as a Booking transaction. Eventually, when the Order is removed from the system, the Force Close logic will close out any remaining Open fact quantities from statement 1 above. What remains in the system will be Shipments and their corresponding pseudo-orders. By construction, the Backlog will be zero.

Transactional/Inventory

Not available for this release.

Transactional/Inventory/Force-zero

Not available for this release.

Pipelined

Use the Pipelined semantic type for facts that can exist in several different life-cycle phases, called pipeline states; for example, sales opportunities or support calls facts.

The fact staging table contains an extra column called *pipe_state*, which serves the special purpose of tracking the position of a given *sskey* in the pipeline of this fact. The designer of this fact table should organize a numbering system for this column in which a larger *pipe_state* number means forward movement in the pipeline. In order to report on this special column, a dimension should be used which tracks changes to the *pipe_state* column (since front-end queries cannot be run against *pipe_state*).

The following logic is used:

- Records with the same *sskey* and *date_key* are filtered using the highest *key* value as described above.
- *sskey* records already in the fact base table are filtered in the same manner as for *Transactional/State-like*. That is, only the latest *date_key* or greater for an *sskey* will survive this filter.
- Only records with *process_key* of 3 (Pipelined) are used.
- For the records that pass filters 1-3, appropriate transactions will be created in the fact base table to place each *sskey* into its proper pipeline stage. When an *sskey* first enters a given pipeline stage, a record will be created for the Booking in that stage. The *transtype_key* is taken from

the fact staging table (recommended range of 4...99). When the same *sskey* subsequently moves forward in the pipeline then a SHIP record (*transtype_key* = 101) will be created in the old pipe state, while a Booking will be created in the new pipe state (again with the *transtype_key* supplied in the staging table). Similarly, backwards movement in the pipeline causes a LOST (*transtype_key* = 3) transaction to be inserted along with a corresponding new Booking.

- A change of dimension within a pipeline stage causes an appropriate delta transaction to be created (the *transtype_key* is taken again from the fact staging table).

005270" 072500

Export/Import of Metadata

All of the control information for an EpiCenter is stored in a single metadata repository called EpiMeta. EpiMeta represents a transactional, fully relational model of over one hundred and fifty tables, with complicated declarative referential integrity constraints. Epiphany provides tools such as Web Builder and EpiCenter Manager for configuring this metadata without the need to write to, or even know about, the underlying data structures.

Epiphany also provides a metadata Export/Import utility for moving metadata between EpiCenters and for backing up the definition of an EpiCenter. To use this tool properly, you should understand the basic metadata concepts discussed in this appendix.

Metadata Overview

All of the user-configurable metadata tables have integer primary keys. These non-natural keys are provided by the database engine when a metadata row is inserted; the value of the primary key itself has no intrinsic meaning. All relationships to this inserted row are made via this integer. For instance, the relationship between a filter group and its filter block is represented in EpiMeta via an integer column called *filter_block_key*.

The Access Export database does not simply contain a copy of the metadata tables being exported for this reason: If data were copied to the Export file, then when this same information is imported into a new EpiMeta, the integer primary key values in the Export file might clash with already existing primary keys in the target EpiMeta. Therefore, the Access database uses an EpiMeta-independent representation of metadata. See *Export File Format* on page 282 for a description of these Access database tables.

EpiMeta contains many tables, all of which are inter-related. Ticksheet metadata refers to constellation metadata (for instance, the attributes refer to dimension columns), while security metadata refers to ticksheets. In order to export only a portion of the metadata at a time, the Export machinery must decide where to stop exporting—otherwise, the entire metadata must be exported with each operation.

When using the Export metadata command of EpiCenter Manager, you must select which part of the metadata to export. Figure H-1 shows an overview of the various domains of metadata within EpiMeta.

Each rectangle in the figure represents one of the options available for export. Everything within the rectangle is actually contained as metadata in the Export file when that option is selected. The arrows in the diagram represent references to other metadata. These references are contained in the Export file as well, but the references are made to other objects by name only. In other words, when exporting ticksheets, the measure mapping metadata is exported in the Export file, but the measures themselves are not exported (unless you also select the Measure option). Only a reference to the appropriate measures is exported.

Upon import, these references are used as follows. When ticksheet metadata is imported into a different EpiMeta, the import machinery searches for measures with the referenced names. If it finds these, then the same relationships are established in the new EpiMeta as the ones that were exported. However, if the Import machinery does not find these measures by name, the measure mapping information will be lost upon import.

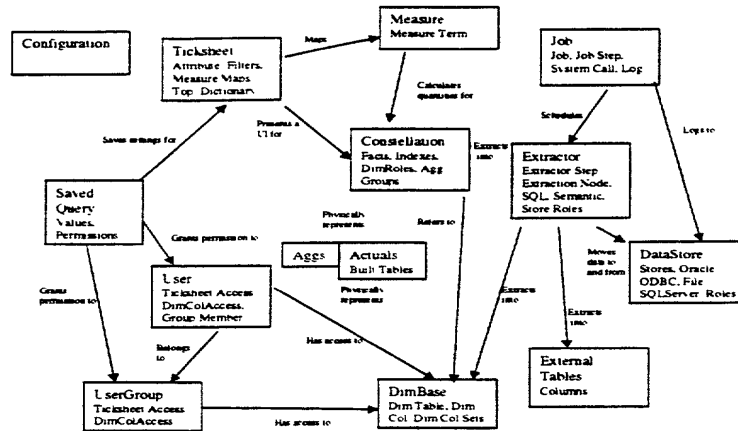


Figure H-1 EpiCenter Data Model

Replacing Existing Metadata on Import

When importing metadata into an existing EpiMeta, the Import machinery will detect an attempt to overwrite existing metadata. The definition of “existing” is usually based on a unique name column for one of the metadata tables. For instance, ticksheets must have unique names, so an attempt to import a ticksheet with the same name as an existing one results in a warning message (unless the Always Replace option has been selected previously).

The Release 3.2 Import machinery uses a Delete/Re-insert model to replace existing metadata. In future versions, a row will be modified “in-place” instead of first being deleted. However, it is important to understand the ramifications of this delete-first operation.

When a row of metadata is deleted, all rows that reference that row are also deleted. For example, suppose a single measure is imported over one with the same name. If ticksheets already refer to that measure, then the Import operation deletes the measure mappings to those ticksheets. First, the measure is deleted (which deletes all the measure mappings to that measure). Then the measure itself is re-imported, but since the Export file does not contain the ticksheet metadata, the measure mappings are not re-imported. Thus exporting metadata and then immediately re-importing it does not produce an equivalent EpiMeta.

Actuals and Agg Metadata

The Export All command in EpiCenter Manager does not export the entire contents of EpiMeta. The options for Actuals and Aggs are omitted by default from this export. This is because these sections of metadata are “derived” metadata: the Schema Generator produces Actuals metadata, and Aggbuilder produces Agg metadata. As a result, a new EpiCenter can be constructed using the Export All metadata by rebuilding the schema, re-extracting, and re-running Aggbuilder.

To “clone” an EpiMeta in such a way that EpiMart itself will not need any modification when it is used with this new EpiMeta, you must select the options for Actuals and Aggs.

Note: An Export All operation on a running system, followed by a re-import of that metadata causes all Actual and Agg metadata to be lost.

Export File Format

Each Microsoft Access Export database has the same schema. In fact, this schema can be thought of as a meta-schema for representing relational data. For a description of these tables, see Table H-1.

To modify the row contents contained in an Export file, edit the *Export_col* table, which is simply a collection of name/value pairs for columns.

Table H-1: Export Tables

Table Name	Description
Export_tbl	One row per metadata table being exported.
Export_row	One row per metadata row being exported.
Export_col	One row per column per row of metadata being exported. Only non-relationship columns are contained in this table.
Export_rel	One row per relationship between two rows of metadata. Can be a relationship between two rows contained in the Export file, or between one row in the Export file and one reference to a row in a foreign EpiMeta.
Export_status	Header information about the Export file.
Rel_parent	A reference to a metadata row in the foreign EpiMeta.

Export/Import of Metadata

09:55:18.072500

Troubleshooting

This chapter describes the Epiphany error conditions and error messages and suggests action you can take to resolve problems.

If you need additional assistance, please e-mail Epiphany Customer Support: support@epiphany.com.

Registry Editor Warning

Using Registry Editor incorrectly can cause serious problems that may require you to reinstall your operating system. Microsoft cannot guarantee that problems resulting from the incorrect use of Registry Editor can be solved. Use Registry Editor at your own risk.

For information about how to edit the Registry, view the "Changing Keys And Values" Help topic in Registry Editor (**Regedit.exe**), or the "Add and Delete Information in the Registry" and "Edit Registry Data" Help topics in **Regedt32.exe**.

Note that you should back up the Registry before you edit it. If you are running Windows NT, you should also update your Emergency Repair Disk (ERD).

SQLServer Error Message

Cannot Connect to the Server

The most likely cause of this problem is that the ODBC driver for SQLServer has not been correctly installed.

You can verify this by opening Start\Settings\Control Panels\ODBC\ and selecting the tab for the ODBC Drivers. Make sure there is an entry for SQLServer. If not, the Microsoft Back Office CD-ROM, Disc 2 CD-ROM, which you used to install SQLServer and its utilities has the ODBC driver in the directory *i386/utils/odbc*.

Application Server Error Messages

The following Application Server errors messages are described:

- *Cannot Connect to the Server* on page 286
- *User Cannot Log In* on page 287
- *Allow Interaction with Desktop Problem* on page 290
- *Out of Memory* on page 290
- *Invalid Object DATE_O Error* on page 291
- *Not a Valid Application Error* on page 292
- *Internal Windows NT Error* on page 292
- *EpiQuery Engine Database Connection Open Failure Exception* on page 293
- *Charts Do Not Display* on page 293
- *GIF Images Fail to Display on Web Pages* on page 297
- *No Results Available for a Query* on page 298
- *Result Page Error: Extraction Date Unknown* on page 298

- *Web Server Message: Object Not Found* on page 299
- *Browser Crashes When Retrieving Results from Application Server* on page 301
- *Refresh Program Fails* on page 301
- *Application Log Full Error* on page 301
- *Application Server Log Security Problem* on page 302

User Cannot Log In

There are three types of failure associated with an unsuccessful login to the Application Server:

1. The Application Server generates a critical exception indicating an application error, or an error related to the NT security domain).
If this case, an error message appears in the browser with a link to the log that contains the stack trace for the error. Read the error description. Generally, it relates to the way NT domain security is configured or set up at this point. For example, the error might say that the domain controller could not be reached.
Try to fix the problem based on the specifics of the error message. If you need additional assistance, please e-mail Epiphany Customer Support: support@epiphany.com.
2. An invalid login message displays even though there is a valid username and password.
This error may result when the search for the username/password occurs on the wrong machine. For example, suppose a user *foo* existed on both the machine local to the Application Server and the primary domain controller.

The order in which the user *foo* is searched is as follows:

- the local SAM
- the primary domain
- the trusted domains

Thus, user *foo* in the local SAM will be found first, even though the username/password combination could have been for the user *foo* that exists in the primary domain or in a trusted domain.

To solve this problem, further specify the username by including the domain name before the username; for example, *EPIPHANY\foo*. In general, specify the full user's name if the user's browser displays an invalid login message.

If the error persists, please e-mail Epiphany Customer Support: support@epiphany.com.

3. Authentication was successful, but the user is not allowed to use the Epiphany system.

For a user to have access to Epiphany System, he or she must be a member of at least one Epiphany group after a sync-up process.

Use EpiCenter Manager to check group memberships after you receive this error message. If the old group memberships were removed, then this error has occurred because in the NT domain, the user is not a member of the NT groups that have the corresponding groups marked Synchronize in EpiCenter Manager.

Also check that the username used to log in matches the username in EpiCenter Manager. If the username in EpiCenter Manager is prefixed with domain name other than the one that the actual NT user is a member of, then the login could fail and return an error message to this effect. Specifying the full username upon log in may fix this problem.

If the error persists, please e-mail Epiphany Customer Support:
support@epiphany.com.

Additional Action to Take If User Still Cannot Log In

Note: If you need to have a working system immediately, you may temporarily disable security by hooking up EpiPassThruLogon module. Be sure to read *Application Server Security* on page 186 for instructions on configuring authentication modules.

If you cannot fix the problem using the above procedures, e-mail a security log (*xxx-xxx-xxx-SECURITY.txt*) and Application Server log (*xxx-xxx-xxx-SRV*) to Epiphany Customer Service.

A description of the three security logs follows:

APPSERVER.LOG	The only file with a suffix of <i>.log</i> , this is the main Application Server activity log, which logs a summary of all queries performed on that server. Includes the user who submitted the query request, the type of ticket submitted, the failure or success of the query, and the amount of time for the query.
SAVE_RESTORE	Logs all save and restore operations during a session.
QM_randomly_generated_alphabetic_sequence_username	Consists of individual logs, one per user, that show the content of each query or other browser transactions. These logs contain the selected attributes, measurement selections, filters, and so forth submitted with each query. It also contains the SQL sequences, the duration times of the various stages of the query, and the total time for each query.

Allow Interaction with Desktop Problem

If you have installed an Application Server and the Service Control Manager (Start Menu\Settings\ControlPanel\Services) fails to start the Application Server, then double-click the service in question, and make sure the Allow Service to Interact with Desktop is selected.

Out of Memory

The Application Server requires sufficient memory (at least 64 Mb) to function properly. The default `jre` program allows the Application Server to allocate only 16 Mb of extra memory. Queries that involve Product and Customer (or any large dimension) and have the Rows option set to All can easily exhaust the default 16 Mb. When this happens, you will receive an Out of Memory exception error: `java.lang.OutOfMemoryError`.

The solution to this problem is to make sure that you have started the Application Server with the `-mx64M` parameter. For example,

```
jre -mx64M -classpath ... com.epiphany.server.Server ...
```

The `-mx64M` parameter allows the Application Server to allocate up to 64 Mb of memory.

The Refresh program can also exhaust memory if run three times consecutively. This is because the atomic switch from old to new metadata requires a least two copies of the metadata to be alive at the same time. Until all references to the old metadata are released, both objects stay in memory. Eventually, as sessions are cleaned up and as old command threads die, the old object will be released.

VirtualMartDatabase Key Missing Error

If you receive this message from the Application Server, more than likely a valid EpiCenter data store has not been specified.

When you configure an EpiMeta database using EpiCenter Manager, you will use the EpiMart Data Store dialog box in the Extraction\Data Stores folder to specify the target EpiMart. Click the Properties tab and make sure that the data store name is correct.

Invalid Object DATE_O Error

When starting the Application Server, you may encounter the SQL error `Invalid object Date_0`. This error is generated when the Application Server attempts to read the metadata (EpiMeta).

To correct this problem:

- Make sure that the EpiMart data store specified via EpiCenter Manager points to a valid EpiMart database. (Open the Extraction\Data Stores\EpiMart Data Store dialog box, and click the Properties tab.)
- Make sure that you have populated the date dimension in your EpiCenter using EpiCenter Manager. To verify this, go to ISQL and use the `sp_help` command on your EpiMart database. Check for the existence of the `Date_O` table. If it is not there, it has not been populated. See *Getting Started* on page 59 for instructions.

Not a Valid Application Error

This problem can occur for two reasons:

- If a service component required for Windows NT, an application, or a network protocol, is corrupted or missing.

To correct this problem, manually expand the service component file. For example, if the *service name* in Event ID 7000 is MUP, expand MUP.SY_ from the Window NT CD-ROM to MUP.SYS in the %SystemRoot%\SYSTEM32\DRIVERS folder.

- If the folder location of the executable contains spaces in the directory name (that is, has a long filename). An example would be when the executable is located in the \Program files\service.exe folder.

To correct this problem, modify the Registry key that contains the executable path so that it is enclosed in quotation marks.

Internal Windows NT Error

This error results when the Epiphany Application Server cannot be started as a service. The exact error message that was returned from the Epiphany Application Server is logged in the Windows NT Event Log.

To locate this error message in the Event Log:

1. Go to Start menu\Programs\Administrative Tools (Common)\Event Viewer.
2. Click the Log menu and select the Application.
3. Double-click the appropriate event.

All Epiphany Application Server events have the source EpiAppServer.

To solve this problem, first stop any running Application Server services. Then log onto Windows NT as an administrator and re-install the Epiphany software. If this does not solve the problem, start the Application Server from the command line as described in *Running as a Console Application* on page 164. The console output should describe what is wrong.

EpiQuery Engine Database Connection Open Failure Exception

An EpiQueryEngineDBConnOpenFailureException from the EpiQueryEngine means that the Application Server is having difficulty connecting to the database. Take these steps to correct this problem:

- Make sure that the username and password for the EpiMeta database are set correctly in the Windows Registry.
- Make sure that the database name and server are also configured properly (again, in the Registry).
- Make sure that SQLServer TCP/IP Sockets have been enabled. Usually, this is option is set when the SQLServer is installed on the machine. To verify this, from the Start menu, open SQLServer 6.5 and select the SQLServer Setup program. Then click Change Network Options and make sure that TCP/IP sockets are enabled. Restart SQLServer if you change any values.

Charts Do Not Display

If there is a broken link to an image, or no image appears, or an image with an error message appears instead of the chart, there is a charting problem. Note the following:

- **makechart.dll** has a log file that logs all calls from browsers to the **isapi.dll**. It is located in the Epiphany system log directory (set by SystemLogDir Registry key) and is called *charts.log*. You can use this file to check whether calls make it through **makechart.dll**. There are two entries for each call, one when the call initiates and another when the call is completed.

Troubleshooting

- **makechart.dll** has the following syntax for calling it (HTML reference):

```
http://bullwinkle/STROMBOLI/  
makechart.dll?makechart?uid=filename.epc&context=STROMB  
OLI&chnum=0&chcollection=1&chtype=4&3d=1&rescale=0&widt  
h=600&height=400&debug=0
```

where:

- **uid** is the name of the EPC (*.epc) file that has data to be charted.
- **context** is the instance name.
- **chnum** is the chart number in the EPC file.
- **chcollection** is the collection number in the EPC file.
- **chtype** is the chart type.
- **3d** is the 3dflag.
- **rescale** is the rescale flag.
- **width** is the width of the image to return.
- **height** is the height of the image to return.
- **debug** is the flag used for debugging.

If there is an error during chart creation in **gsmaker.exe**, **makechart.dll** generates an image with an error message in it. If the debug verbosity level is set to 0, the error message is Charts are not available. If debug is set to 1, the error message should be more informative. It may describe a problem with an EPC file, or show an exception.

Before attempting to diagnose any charting problem:

1. Try to isolate the part of HTML (the URL) that calls charts, and enter it into the URL address box in the browser window.
2. View the source on the page that includes charts.

3. Find the part of HTML that calls **makechart.dll** and paste it into the URL address box.
4. Change the debug flag from 0 to 1.
5. Go to that Web page.

If there is a broken link to an image or no image appears, the problem could be caused by **makechart.dll** not being called, or **gsmaker.exe** looping or crashing. Possible causes for this problem are:

- **makechart.dll** is not installed correctly. Check the *WWWROOT* directory for the instance for **makechart.dll**. Make sure it is present. See *Manual Chart Install* on page 220 for instructions on installing **makechart.dll**.
- **makechart.dll** has different parameters from the HTTP request actually being made to it. Compare the actual HTML request with the HTML syntax for calling **makechart.dll** given above. If the two are different, make sure that your Application Server and **makechart.dll** derive from the same build of the product.
- **gsmaker.exe** has an application error that put it in an infinite loop or caused it to crash. Check to see if **gsmaker.exe** is consuming all the CPU's resources. If it is, then kill the program, or reboot the machine. Check for an existence of **GSW32.exe** process. This is the graphics server package engine. It should go away after a chart is successfully created. If the process is still running, that means a severe failure has occurred. If it seems that **gsmaker.exe** is not functioning properly, e-mail the EPC file used to generate this chart and the **makechart.dll** HTML reference to Epiphany Customer Service.

If an error message is returned instead of a chart, your problem could be the result of one of these circumstances:

If an error is COM (Common Object Model) related, check for the following errors. COM error codes start with 800 and are most likely caused by incorrect or incomplete setup.

- Not enough storage is available to complete this operation.

This error indicates **gsmaker.exe** terminated during creation of the chart. E-mail the EPC file used to generate this chart and the **makechart.dll** HTML reference to Epiphany Customer Service.

- Access is denied.

This error indicates **gsmaker.exe** is not properly installed. Use **dcomcnfg** to determine the special group that Everyone has access to and to launch permissions to the EpiChart Class COM object. See *Manual Chart Install* on page 220 for more information.

- If an error is **gsmaker.exe** related (begins with GSMaker error), then check for these errors:

- If there is an EPC file that is related, then the Application Server created a bad .EPC file. E-mail the EPC file and the **makechart.dll** HTML reference to Epiphany Customer Service.

- If the error says could not initialize graphics server subsystem, check that Graphics Server files have been installed. (The Graphics Server consists of two parts: **makechart.dll**, which should never have any problems and **gsmaker.exe** and its auxiliary files, which could be installed incorrectly or configured poorly by the installation program.) If these files are there, reboot the computer, run the same query and the chart should show up. If this occurs, then **gsmaker.exe** is slowly leaking resources. Report this problem to Epiphany Customer Service.

If no chart is returned and a dialog box on the Application Server machine that indicates the Initialization of the dynamic linked library **user32.dll** failed, and the process is terminating abnormally, then **gsmaker.exe** is not properly installed.

Use **dcomcnfg** to determine if EpiChart Class is running under the Identity of the query user. See *Manual Chart Install* on page 220 for more information.

If the problem persists, e-mail Epiphany Customer Service.

GIF Images Fail to Display on Web Pages

If GIF images do not appear on your Web pages, do the following:

1. Make sure that your Web server has an alias for your *instance_name* that points to a valid directory. If you performed a normal installation, then all Web files should be located in the directory:

C:\Program Files\Epiphany\Instance_name\web\WWWROOT

and GIF files should be located in the directory:

C:\Program Files\Epiphany\Instance_name\web\WWWROOT\images

(assuming your Epiphany Application Server was installed in *C:\Program Files*).

If your Web server is IIS 3.0, you can find the aliases by opening up the IIS Internet Service Manager (normally this is located in your *Start\Microsoft Internet Information Server* menu) and selecting the Directory tab. Make sure that there is an entry for *instance_name* that has a valid directory.

2. Make sure the *WWWROOT\images* directory has the GIFs in it. If you are missing a few GIFs, then you will need to reinstall your Application Server or copy the GIFs from a different instance.
3. Check the BASE HREF tag that is defined in the source of the page. In your browser, try to view the source for this HTML page. Look for the BASE HREF tag at the top of the page. Note what it is and make sure that it is a valid alias using the procedure above.

4. Make sure your Web server is serving pages and that your browser is not displaying cached HTML. Clear the caches (Memory and Disk) on your browser, close your browser, and try to access the URL again. Also, try referencing another URL from your Web server to make sure that it is running.

Technical Note: All of the GIFs on an Application Server-generated page are referenced from the *images* directory, which is relative to the BASE HREF specified at the top of the page in a META tag. The *instance_name* is derived from the URL that you use to access the Application Server. It is used throughout the system to read the correct Registry entries and to generate the correct URLs.

No Results Available for a Query

The most common cause of this problem is that the *current_datamart* key in the EpiMeta database is set to the wrong database.

Use EpiCenter Manager to make sure that the *current_datamart* is set to the correct database. You can also use the ISQL tool to determine which database, A or B, has the most current data.

Result Page Error: Extraction Date Unknown

The *last_extract_date* is a field that is kept in the EpiMeta database. It is used to keep track of the date displayed on the top of all Clarity and Relevance reports as the date of the last extraction. It is normally populated by the extraction SQL entered in the End of Extraction job in the EpiCenter Manager. It can also be populated by EpiCenter Manager via the Configuration dialog box. This field must be entered in one of two very strict formats. The default format is *mm/dd/yyyy*; for example: 01/14/1998.

The Application Server applies the following logic to parse the date:

1. If the field has more than 10 characters, then parse it using the pattern *mm/dd/yyyy hh:mm:ss*. Otherwise, use the pattern *mm/dd/yyyy*.
2. If the parse fails, then use {extraction date unknown}.

In addition, the date that is displayed at the top of the report always has a time zone. The time zone is printed based on the default time zone of the machine on which the Application Server is running. The date that is being displayed is also taken into consideration. For example, the date 12/20/1997 will display as December 20, 1997 PST if the Application Server was running on a machine in California. However, the day 05/12/1998 will display as May 12, 1998 PDT on the same machine since Daylight Savings time took affect in April.

Note: EpiCenter Manager does not allow the user to enter a date in the format `mm/dd/yyyy hh:mm:ss`. Only the SQL in the extraction job can enter dates of this format, or you can manually arrange this via ISQL.

Web Server Message: Object Not Found

If you installed the Epiphany software using the standard Epiphany software installation program, you will access your Web server through this type of URL: `http://machinename/scripts/instance_name/Epiphany.dll`.

If you receive an object not found error message, follow these steps:

1. Start and stop the Web server. If this does not solve the problem, go to the next step.
2. Verify the Web server is serving pages.

Note: Make sure that your browser is not just returning cached HTML pages by clearing your memory and disk cache before testing.

Try to access other URLs from the same *machinename*. Try to access other static HTML files that are installed as a part of the Application Server installation, such as `http://machinename/instance_name/clarityhelp.html`.

3. If this does not work, try accessing any other file that the Web server should be serving. Consult the Internet Service Manager for the names of other aliases that the Web server should be serving, and then try to access these aliases with your browser.

In most cases, your Web server searches the *C:\inetpub\scripts\instance_name* directory to find the **Epiphany.dll** program. Make sure that there is such a directory on your machine, and that the **Epiphany.dll** file is in that directory.

4. Check the file permissions for the **Epiphany.dll** file.

First, make sure that the account IIS uses for anonymous logins has file access permissions for the **Epiphany.dll** file. Go to the IIS 3.0 Internet Service Manager and look at the Anonymous Login account box. In IIS 4.0, right-click the name of the machine and choose Properties. Select the Directory Security tab. In the Authentication Methods dialog box, select Allow Anonymous Access and click the Edit button to modify the account used for this purpose. Make sure the user and password are correct.

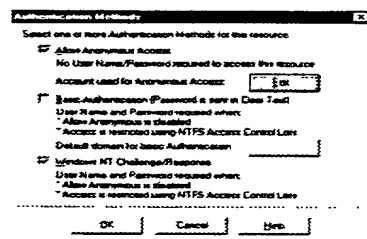


Figure H-1 Authentication Methods Dialog Box

To check file permissions, open the Windows NT Explorer window and right-click the **Epiphany.dll** file in the *./scripts/instance_name* directory specified above. Go to Properties and open the Security tab. Click View Permissions and make sure that the account that you used for Anonymous Web Login has access to this file. (If Everyone is selected, then all people, including the Web login account, have access to the file.)

Browser Crashes When Retrieving Results from Application Server

In general, for machines with less than 32 Mb of RAM, the browser will perform very poorly when parsing HTML files that are larger than 150 K. Therefore, users who do not have at least 32 Mb of RAM installed on their machines should refrain from retrieving large queries.

An example of a large query follows. Suppose you query Customer by Fiscal Year and apply the filter Business Unit: Learning. This query returns approximately 3800 customers, and the HTML that is generated is 1.8 Mb. When loaded in Netscape 4.03, it occupies 37 Mb, takes 3 minutes to parse, and consumes the entire processor. The parsing is the bottleneck in the downloading of the file.

Note: When started, Netscape Navigator 4.03 requires 8 Mb immediately to load whatever it is loading. Internet Explorer (IE) requires 6.7 Mb, and is substantially faster at parsing the text.

Refresh Program Fails

If the Refresh program fails, the Application Server will continue to use the old metadata information. Users will still be able to log in and view the old ticksheets. This is because the Refresh program reads the new metadata into temporary structures that are atomically exchanged with the old structures only if all of the refresh structures were read correctly.

To solve this problem, rerun the Refresh program.

Application Log Full Error

If you receive this error, you can take the following action to delete all log events.

Note: If you do not want to delete all of the log events, you can also increase the amount of space allocated to the Application Log. You can also select the Override as Needed option in the Log Properties dialog box.

Troubleshooting

1. Open the Start\Programs\Administrative Tools (Common)\Event Viewer. Select the Application Log under the Log menu item.
2. Select Clear All Events from the Log menu.

When prompted whether you want to save the log files, and whether you really want to delete all of the log events, respond in the affirmative.

Application Server Log Security Problem

The *APPSERVER* and *SRV* log contain all of the session IDs that have logged in. Although unlikely, it is possible that someone could alter a session ID (along with a fake IP address) and run queries against the Application Server.

The log file also contains information about who is running a query, and when they are running a query, which may be confidential information.

The log files for each query contain all the information necessary to rerun a query. Hence, a malicious party can also determine the type of queries run by a particular user.

It is important to give user access to the log file directory so that when problems occur, the Download query log link at the bottom of each results page will work. Directory browsing for the log file directory, however, should be turned off.

To turn off directory browsing, de-select the Directory Browsing Allowed option in the Directories tab of the WWW Services Properties dialog box in the Internet Service Manager. (Normally, IIS is located in your *Start\Microsoft Internet Information Server* menu.)

GLOSSARY

Actual Tables

The Actual tables are metadata created by the Adaptive Schema Generator after it has built or modified tables in EpiMart. The Actual tables are consulted on subsequent schema generations to determine delta operations to perform. Also, other tools examine these tables as a check to ensure that EpiMart's schema matches the current metadata definition.

Adaptive Schema Generator

The Adaptive Schema Generator is a component of EpiCenter Manager that builds the tables in EpiMart using the schema metadata definitions in EpiMeta. When schema metadata changes, this program can perform delta operations to modify the schema without losing data in EpiMart.

Aggregate Builder (Aggbuilder)

Aggbuilder is the executable program (**agg.exe**) that builds aggregate tables in EpiMart. Normally, the execution of this program is scheduled after all data has been extracted from source systems and merged into EpiMart.

Aggregate Group

An aggregate group is a metadata definition of the fact aggregates to be built. It is a series of instructions that tell Aggbuilder which aggregates to build on one or more fact tables. An aggregate group applies to a single constellation. The actual fact aggregates that get built are determined by a combinatorial expansion of the instructions in the aggregate group.

Aggregate Navigation

Aggregate navigation is the process by which the Epiphany query machinery determines the optimal aggregate table to use to satisfy an application's request for information. Applications such as Clarity do not need to know about the presence of aggregates because the aggregate navigation process makes the aggregate layer abstract.

Application Server

The Epiphany Application Server is the Windows NT Service that connects with a Web Server and serves up Epiphany ticksheets and reports. An Application Server is configured to connect with an EpiCenter.

Attribute

An attribute is a dimension selection on a ticksheet. The row and column lists in Clarity contain attributes. Attributes are related to exactly one dimension column in a table. The only difference between an attribute in a table and one in the Epiphany system is that an Epiphany attribute has an associated display label, such as Fiscal Year, that can be configured via Web Builder.

Attribute Role

An attribute role is the usage of an attribute on a ticksheet. Attributes are defined only once per ticksheet. If you want a single attribute to appear in both the rows and columns listboxes of Clarity, use Web Builder to assign that attribute both of these roles: Clarity Row and Clarity Column.

Backlog Type

Backlog type is used on a measure term to specify that the measure term should exhibit accumulation behavior. When time is one of the dimensions of a query, the results for different time periods will exhibit beginning or ending accumulated values instead of the actual transactions that occur in that time period. BEGIN specifies beginning accumulated value, whereas END specifies ending accumulated value.

Base Dimension

A base dimension is a physical dimension table in EpiMart, such as Customer and Product. Base dimension tables contain the actual dimensional values that are available for reporting or filtering. Base dimension tables can be used more than once in a single constellation via dimension roles. Base dimensions are defined for the entire EpiCenter and can be shared by all of the constellations within the EpiCenter.

Base Dimension Aggregate

A base dimension aggregate is a physical table in EpiMart that represents an aggregated view of a base dimension base table. A base dimension aggregate results from the removal of one or more columns from the base table, followed by the removal of duplicate rows caused by this deletion.

Base Table

Base tables are the EpiMart tables (both fact and dimension) that store non-aggregated customer data at the level of granularity of extraction. These tables will be used as input to Aggbuilder to build aggregate tables with the same information at different levels of granularity. Base table names end with an underscore zero (_O); for example, Order_O_A or Order_O_B.

Constellation

A constellation is a grouping mechanism for like-structured fact tables. A constellation defines the dimensionality of all fact tables in that constellation. Dimension roles and degenerate dimensions are defined on a constellation, not a specific fact table. All fact tables placed in that constellation then inherit that definition. Both ticksheets and measures are defined within the scope of a single constellation, and thus apply only to the dimensionality defined by that constellation.

Custom Index

A custom index is the metadata definition of indexes to build on fact tables in EpiMart. Normally, each fact table is given a single index. You may use EpiCenter Manager to specify additional indexes to build. You must use the semantic template Custom Fact Index to actually build the indexes.

Data Store

A data store is a logical location of data to be used either as a source or sink within an EpiCenter. Data stores include relational database connections, and also files and directories.

Dataset

A dataset is a logical user-interface grouping of ticksheets that display the same view of data. Typically, different ticksheets types (such as Clarity and Relevance Profiling) that show the same data within a constellation are placed within a dataset. The dataset appears to end-users as a named entity that corresponds to a single business process. For instance, a dataset called *Executive Summary* might be defined that shows similar amounts of high-level data, using different Epiphany applications.

Date Dimension

The Date dimension is a special base dimension table supplied by Epiphany, which is used for storing all attributes related to time. All fact tables in an EpiCenter receive the foreign key *date_key* to this table.

Degenerate Dimension

A degenerate dimension is a column in a fact table that stores textual information in lieu of using a foreign key to a base dimension table. Degenerate dimensions are used when a single field of data fully specifies a dimension of that fact. Examples include Invoice Number and Serial Number.

Because degenerate dimensions appear only in fact base tables, they are never aggregated. (They are, however, always indexed.) Degenerate dimensions are defined on a constellation rather than for a specific fact table. All fact tables within that constellation inherit the degenerate dimension definition.

Dictionary Entry

A dictionary entry provides end users with online definitions of the terminology used on a ticksheet, such as Units, Gross, and Sell-Through, which are hyperlinked to a dictionary page. The creator of the ticksheet configures these entries via Web Builder.

Dimension Column

A dimension column is a single column or attribute of a base dimension table. It contains the actual customer data that appears on reports or in filters.

Dimension Column Access

Dimension column access is the ability for a user or group to view only a subset of the available data in an EpiCenter. For example, dimension column access settings can be used to limit specific users to only data for the Western Region.

Dimension Column Set

A dimension column set is a named set of dimension columns (all within a single base dimension) that defines which columns will be used in a base dimension aggregate.

Dimension Role

Dimension roles allow a single base dimension table to be used in different roles within a constellation; for example, a Territory base dimension table includes Eastern Sales and Western Sales data. A Western Sales dimension role would reference the Territory base dimension for its data.

A dimension role is a dimension column in a fact table that defines the foreign key that references a base dimension table. It always has an associated base dimension table. Dimension roles are defined within a constellation, and all fact tables in that constellation inherit the dimension role. Multiple roles within a single constellation can refer to the same base dimension.

EpiCenter

An EpiCenter is a single logical Epiphany database installation that includes customer data in addition to control metadata. It physically consists of two databases: EpiMeta and its associated EpiMart.

EpiCenter Enterprise Manager (EpiCenter Manager)

EpiCenter Enterprise Manager is a Microsoft Windows program for configuring EpiCenters. These key components of the Epiphany system are available via EpiCenter Manager: schema, extraction, and security metadata, the Adaptive Schema Generator, and metadata export/import.

EpiChannel

EpiChannel is the Epiphany program (**extract**) that executes extraction jobs. The person who runs EpiChannel enters parameters to connect to an EpiCenter and specifies jobs to be executed. EpiChannel then implements the extraction steps declared as metadata in the EpiCenter.

EpiMart

The physical database that contains actual customer data. The schema of tables in EpiMart is determined by the metadata in EpiMeta. The data contents of EpiMart tables are determined by the extraction steps in EpiMeta.

EpiMeta

EpiMeta is the metadata repository for an EpiCenter. It contains all control information for the EpiCenter and therefore defines the behavior of that EpiCenter. EpiMeta points to EpiMart via the special EpiMart data store, which is available in every EpiMeta.

External Column

An external column is a single column in an external table.

External Table

An external table is a table built in EpiMart, usually as a temporary table used during extraction. External tables must be declared as metadata (as opposed to begin built outside of EpiPhany) in order for the Adaptive Schema Generator to know of their existence. Otherwise, the table would be purged by the Purge EpiMart command.

Extraction Group

An extraction group is a set of extraction steps.

Extraction Step

An extraction step is a single atomic extraction operation. It can be either a SQL statement or a semantic Instance.

Extractor

An extractor is a list of extractor steps together with input and output data stores. It logically specifies a series of steps that move data from the input to the output data store.

Extractor Step

An extractor step is a single step of an extractor, which always points to an extraction group. In this way, extractors consist of an ordered list of extraction groups (which are ordered lists of extraction steps).

Fact Aggregate

A fact aggregate is a physical table in EpiMart that contains aggregated fact information for a single fact table.

Fact Column

A fact column is a single numeric column in a fact table.

Fact Table

A fact table is a physical table in EpiMart that contains numeric data in addition to references to dimension tables that specify attributes about the fact, such as who, what, when, and where the fact occurred.

Filter Block

On a ticksheet, a filter block controls the user's ability to filter on a single dimension column (for a dimension role of the ticksheet's constellation). The filter block also defines the appearance of the filter (for example, check box versus listbox).

Filter Element

A filter element is a single check box for filtering within a filter group, or a single entry within a listbox filter block.

Filter Group

A filter group is a logical grouping of filter elements within a filter block. It is applicable only to check box filter blocks.

Job

A job is a top-level workflow object that defines a sequence of steps to be performed by EpiChannel. It also specifies the logging locations for that execution.

Job Step

A job step is a single step of a job, which can be either an extractor or a system call.

Measure

A measure is a single business calculation. It can be an arithmetic combination of measure terms using RPN (Reverse Polish Notation).

Measure Mapping

The mapping of ticksheet selection columns to measures.

Measure Term

A measure term is a single component of a measure. It refers to the aggregation of a single fact column, such as *SUM(Order.net_price)*, with a particular transaction type. It can be combined with other measure terms to create a composite measure (business calculation).

Pull/Push SQL Statement

A pull/push SQL statement is a type of extraction step that issues SQL against an input data store and then pushes the result set into a table in the output data store of an extractor.

Query Machinery

The component of the Epiphany Application Server that communicates with EpiMart and actually issues SQL statements against the DBMS. The query machinery is a common component of Epiphany's front-end applications.

Report Gallery

The Report Gallery is a component of Epiphany's front-end user interface that allows users to view Saved Queries.

Saved Query

A Saved Query is a saved set of ticksheet settings from a previous query that can be viewed again. Users and groups are granted permission to use or modify Saved Queries. Users who request the same Saved Query (and have the same dimension column Access rights) will receive the same output (reports, charts, and so forth).

Selection Column

A selection column is a single column in the measure section of a ticksheet. The combination of one value from each selection column translates into a single measure in the report output.

Selection Column Element

A selection column element is a single value in a selection column. For instance, a selection column might contain the elements *Gross*, *Net*, and *Return*.

Semantic Instance

A semantic instance is a post-compilation SQL program. When a semantic template is applied to either a base dimension or fact table, the template becomes instantiated as an actual SQL program that can be used to accomplish specific business rules during extraction.

Semantic Template

A semantic template is a generic SQL program intended to accomplish specific business rules (these rules are referred to as semantic types) during extraction. A semantic template does not refer to actual column or table names. Only when the template is applied to a base dimension or fact table (via a semantic instance) does the SQL contained within it refer to real column and table names.

09625518.072500

Semantic Type

A semantic type refers to the logical business process for which the semantic template is applied.

Source System

A source system is the logical notion of a source of business data. Two physical databases (one a backup of the other) might represent the same source system within Epiphany.

SQL Statement

A SQL statement is an extraction step that issues custom SQL against an input data store. The results of the SQL statement can either be discarded or used to push data into a table in the output data store.

Stand-Alone SQL Statement

A stand-alone SQL statement is a SQL statement whose results are discarded.

System Call

A system call is an extraction step that causes an operating system program to be invoked.

Ticksheet

A ticksheet is a form that end users open in a Web browser and use to submit queries to the data warehouse. It is called a ticksheet because users make selections by ticking (selecting) items.

The ticksheet developer uses the Ticksheet dialog box in Web Builder to construct these ticksheets.

Transaction Type

Transaction types distinguish rows with different interpretations in the same fact table. For example, an Order fact table might contain both a BOOK and SHIP transaction type, differentiated by the value of the column *transtype_key*.

Measure terms created via Web Builder specify a transaction type in addition to a fact column. This allows the summation of only those rows in a fact table with the same transaction type.

Web Builder

Web Builder is a Windows-based program developed by Epiphany for configuring user-interface metadata, including measures, ticksheets, and dictionary entries.

09625518.072500



INSTALLATION GUIDE

3.4

005270" 8T552860

January 1999

Copyright and Trademarks

Copyright © 1998-1999 by Epiphany Marketing Software, Inc. All rights reserved.

This document and the software it describes are furnished under license and may be used or copied only in accordance with such license. Except as permitted by such license, the contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, without the prior written permission of Epiphany Marketing Software, Inc.

This document contains propriety and confidential information of Epiphany Marketing Software, Inc. The contents of this document are for informational use only, and the contents are subject to change without notice. Epiphany Marketing Software, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

Unpublished rights reserved under the copyright Laws of the United States.

Clarity™, Relevance™, Momentum™, and Magnitude™ are trademarks of Epiphany Marketing Software, Inc. All other products or name brands are trademarks of their respective holders.

Printed in the USA

Restricted Rights Legend

Software and accompanying materials acquired with United States Federal Government funds or intended for use within or for any United States federal agency are provided with "Restricted Rights" as defined in DFARS 252.227-7013(c)(1(ii)) or FAR 52.227-19.

09625518 072500

09625518.072500

CONTENTS

INTRODUCTION	5
About Epiphany	5
About Release 3.4	5
About This Guide	6
How To Use This Guide	6
Before You Begin	7
 CHAPTER 1	
PREPARING TO INSTALL EPIPHANY SOFTWARE	9
Order of Installation	11
Installing and Configuring Your EpiCenter Host Computer	11
Installing and Configuring a Windows NT Server Host	12
Hardware Requirements for Windows NT Server	12
Windows NT Server 4.0 Installation	13
Disk-Volume Configuration	15
Network Connectivity	15
Installing and Configuring a Solaris Host	16
Hardware Requirements for Solaris	16
Solaris Installation	18
Configuration of Solaris Resources	18
Disk-Volume Configuration	22
Network Connectivity	22
Installing and Configuring Your Database Server	23
Installing and Configuring SQL Server	23
Installing SQL Server	23
Configuring SQL Server	24
Assigning Data Stores in SQL Server	27
Creating Databases in SQL Server	28
Configuring the Temporary Database	28

Installing and Configuring Oracle	29
Installing Oracle	29
Configuring Oracle	31
Creating Control Files	33
Assigning Data Stores in Oracle	34
Creating Users for EpiMart and EpiMeta Tables	35
Preparing Your Application Host	35
Installing Windows NT Server 4.0 on Your Application Host	36
Installing Internet Information Server, Version 4.0	36
Installing Microsoft Office Components	37
Installing and Configuring Connectivity Packages for Your Database Client	
Software	37
SQL Utilities	37
Oracle8 Client	37
 CHAPTER 2	
INSTALLING EPIPHANY SOFTWARE	39
Performing the Installation	41
Application Server	42
Remote Administration	44
Configuring Epiphany Software	45
Configuring the Epiphany Web-server Proxy	45
Setting Up NT Performance Monitoring for Extraction	46
Verifying Your Installation	47
 CHAPTER 3	
MIGRATING FROM RELEASE 3.2 OR 3.3	49

INTRODUCTION

ABOUT EPIPHANY

Epiphany is the leading provider of Relationship Management (ERM) applications. Epiphany provides complete solutions for managing customer relationships by providing Web-based access to relevant information that is captured by various departments within a business organization. Epiphany applications allow you to browse through this information, gain new insights, and explore relationships and trends that might otherwise remain hidden within your various databases.

With packaged applications from Epiphany, you can:

- Browse current data from throughout your organization
- Drill down to analyze specific situations in detail
- Act on that information to satisfy current customers and cultivate new ones

ABOUT RELEASE 3.4

Epiphany Release 3.4 provides the following features and enhancements:

- Support for larger data sets than in previous versions
- Support for the following database-server platforms:
 - Oracle8[®] on Solaris[®] 2.6
 - SQL Server[®] 7.0 on Windows NT Server[®] 4.0

For best performance with SQL Server 7.0, Epiphany recommends that you install the Enterprise Versions of both Windows NT Server and SQL Server.

ABOUT THIS GUIDE

This manual is intended for database administrators and consultants who install and configure EpiCenter data marts and Epiphany applications. The installation procedures that are documented in this guide take between two and three hours to complete.

If specific instructions do not appear for a particular configuration step or option, default values are acceptable.

HOW TO USE THIS GUIDE

Epiphany provides on-line versions of our manuals in PDF form, with a full-text search index that you can use to locate keywords of interest to you. You can open and view these manuals with Adobe Acrobat Reader. However, if you want to use the keyword index to search across Epiphany manuals, you must use Acrobat Reader with Search or Acrobat Exchange.

You can download the free Acrobat Reader with Search from the following Adobe Web site: <http://www.adobe.com/prodindex/acrobat/readstep.html>. After you register with Adobe, select Acrobat Reader with Search from the pop-up menu, along with your language and platform, then click download and follow the instructions provided by Adobe.

After you have installed Acrobat Reader with Search, you can follow these steps to perform a keyword search:

- Step 1:** Open the PDF file for an Epiphany manual in Acrobat Reader with Search.
- Step 2:** From the main menu, choose Tools, then Search, and then Query.
- Step 3:** In the Find Results Containing text box, enter the keywords that you want to search for in the Epiphany documentation set. You can include multiple keywords and wild cards (an asterisk for multiple characters and a question mark for a single character).

Step 4: Click Search.

The documents that contain text that matches your search query are listed in rank order in the Search Results window.

Step 5: Double-click a document in the Search Results list to see the first occurrence that it contains.

Step 6: Use the Search Next and Search Previous buttons in the toolbar to navigate to other occurrences.

For further instruction on using Acrobat Exchange or Acrobat Reader, refer to the on-line guides for Acrobat Reader, which are available from the Help menu.

BEFORE YOU BEGIN

Please follow these steps before you install Epiphany software:

Step 1: Read Chapter 1 of this guide to ensure that you understand the following topics:

- The considerations involved in preparing the host computer, the operating system (OS), the database server, and networking connectivity for your EpiCenter datamart
- The steps that are required to prepare the host computer for your Epiphany application software

Step 2: Make sure that you have the appropriate installation and configuration manuals on hand for the hardware and software products on which your Epiphany software depends. Epiphany suggests that you obtain the following manuals, depending upon the database-server option you select.

- *Net8 Getting Started* (for use with Windows NT Server clients connected to Oracle/Solaris datamarts)
- *Oracle8 Installation Guide for SunSPARC Solaris*
- *Oracle8 Reference*

Before You Begin

- *Solaris 2.6 Hardware, SMCC Hardware Platform Guide*
- *Solaris 2.6 SPARC Installation Instructions*
- *SQL Server Administration Guide* (for Windows NT Server)
- *Start Here: Basics and Installation, Microsoft Windows NT Server*

Step 3: Please install the hardware and software products that are mentioned in this guide according to manufacturer instructions.

Step 4: Please follow the configuration recommendations that are suggested in this guide.

09625512 072500

PREPARING TO INSTALL EPIPHANY SOFTWARE

The process of installing Epiphany software requires that you first install and configure hardware and software components that are supplied by other manufacturers. The success of your Epiphany application depends on the correct installation and proper configuration of these components.

This manual provides general installation guidelines and configuration recommendations for the products on which Epiphany software depends. For specific instructions about a particular product, please refer to the manufacturer's documentation. The configuration recommendations discussed in this manual apply to Epiphany enterprise-relationship-management (ERM) applications only.

The components that you must install and configure before installing Epiphany software include:

- A dedicated host computer for your EpiCenter datamart (strongly recommended) running one of the following operating systems:
 - Windows NT Server 4.0 (Epiphany recommends the Enterprise Edition) with Service Pack 3 and the Microsoft Exchange mail-client utility from Options Pack 4
 - Sparc Solaris 2.6
- RAID disk-volume support (optional but recommended)

09625518-072500

Preparing to Install Epiphany Software

- A relational database server from the following list:
 - SQL Server 7.0 (Epiphany recommends the Enterprise Edition)
 - Oracle8, Release 8.0.5 for Solaris
- A host computer for your Epiphany application running Windows NT Server
- Client utilities for your application host, including:
 - Microsoft Internet Information Server (IIS), Version 4.0, and the Microsoft Exchange Mail utility (available with Option Pack 4 for Windows NT Server)
 - The appropriate connectivity and management package:
 - SQL Server Utilities with a datamart that resides on SQL Server
 - Oracle8 Client for Windows with a datamart that resides on Oracle

Note: A separate application host is required only when you install your EpiCenter datamart on a Solaris host. If you install your datamart and Epiphany application software on the same computer (running Windows NT Server), you must install the appropriate client utilities listed above on your datamart host.

ORDER OF INSTALLATION

Figure 1 illustrates the order in which you install and configure the components of your Epiphany application.

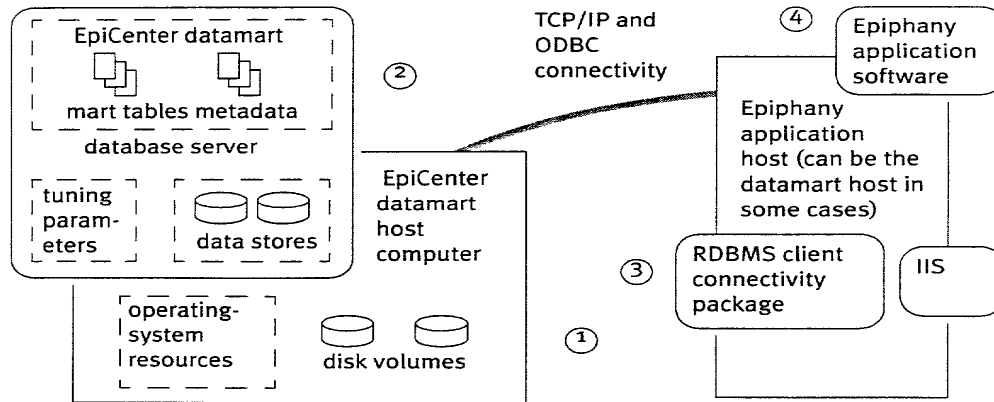


Figure 1: Epiphany Installation Components

INSTALLING AND CONFIGURING YOUR EPICENTER HOST COMPUTER

The specific installation and configuration tasks you must perform depend upon the host computer on which you intend to install your EpiCenter datamart. The sections that follow discuss the installation and configuration process for the following hosts:

- Pentium-based computers running Windows NT Server 4.0
- Sparc-based computers running Solaris 2.6

Installing and Configuring a Windows NT Server Host

This section provides guidelines for installing and configuring a computer running Windows NT Server 4.0 as an EpiCenter host. See “Installing and Configuring a Solaris Host” on page 16 for information about Solaris hosts.

Hardware Requirements for Windows NT Server

Please ensure that your computer meets the following minimum requirements:

- At least one CPU with a speed of 200 megahertz or higher (more CPUs are preferable), capable of supporting Windows NT Server 4.0

Refer to the Microsoft manual entitled *Hardware Compatibility List; Microsoft Windows NT*, Version 4.0, for listings of computers that support Windows NT Server. For enhanced performance, Epiphany recommends that you choose a computer that supports the Enterprise Edition of this OS.

- A minimum of 128 megabytes of random-access memory (RAM)
- Swap space of at least three times the size of RAM.
- The computer must be connected to a network and have TCP/IP network controller installed. Multiple network-card configurations are not currently supported.
- An adequate amount of disk space:

Epiphany software requires about 30 megabytes of disk space (including supporting software such as ODBC and JDBC drivers). The size of your data warehouse depends on the amount of data that you extract from your source systems. If you already have specifications for your datamart, you can use that information to estimate the size of your datamart data, taking into account the following considerations:

- The datamart includes two copies of the database and aggregates that are between three and ten times the base-table size, with additional space needed for staging tables, indexes, keys, and 100 megabytes of Epiphany metadata.
- Epiphany recommends that you reserve a separate log volume that is about 10 percent of the size of your datamart.

Installing and Configuring a Windows NT Server Host

- Epiphany recommends that you reserve a volume that is at least twice the size of your largest fact table for temporary tables.

Based on these considerations, Epiphany suggests that you reserve the following disk volumes for your EpiCenter datamart:

- a RAID-1+0 volume for your EpiMart data and EpiMeta metadata

Use the following formula to determine the size of this volume:

$$\text{mart_vol_size} = 2 * (\text{aggs} * \text{fact_rows} * (100 + \text{fact_row_width})) + 2 * (\text{dimension_rows} * (20 + \text{dimension_row_width})) + 400 \text{ Mb}$$

Replace *aggs* with a value between 3 and 10 depending on the degree to which your application requires aggregate data. Use a higher value for applications that require extensive use of aggregate data.

- a RAID-1 volume for the log device associated with your datamart

$$\text{log_vol_size} = (0.1 * \text{mart_vol_size}) + 10\text{MB}$$

- a RAID-5 volume for the temporary database

Use the following formula:

$$\text{temp_db_size} = \text{tmp_factor} * \text{fact_rows} * (100 + \text{fact_row_width})$$

Replace *tmp_factor* with a value between 2 and 10, depending on your application. Use a value of 2 if you plan to install the Magnitude option.

The Magnitude option requires two additional RAID-0 volumes for external sort space. Use the following formulas for these volumes:

$$\text{sort_working_vol} = \text{fact_rows} * (100 + \text{fact_row_size})$$

$$\text{sort_temp_vol} = 10 * \text{fact_rows} * (100 + \text{fact_row_size})$$

Windows NT Server 4.0 Installation

You must install Microsoft Windows NT Server 4.0, Service Pack 3.0 (or higher), and selected Microsoft Office components. For enhanced performance, Epiphany recommends that you install Enterprise Edition of Windows NT Server 4.0.

Installing and Configuring a Windows NT Server Host

Installing Windows NT Server

To install Windows NT Server 4.0, follow the directions provided in Part 2 of the Microsoft manual entitled *Start Here: Basics and Installation, Microsoft Windows NT Server*, Version 4.0, and follow these recommendations:

1. Select the default directory location in which to install Windows NT Server.
2. Assign a computer name of 15 or fewer characters and write this name down for future use.
3. Indicate the server type as a stand-alone server.
4. Enter your administrator account name and password.
5. Create a repair disk in case of an emergency.
6. Select the default list of components.

As part of the Windows NT Setup, after you finish setting up the network parameters, the Finishing Setup dialog box is displayed, which informs you that are about to start setting up Microsoft Internet Information Server, Version 2.0. Exit the installer at this time. You do not need to install IIS, Version 2.

Note: When installing the operating system, you might have to install additional drivers for 3rd-party disk drives, graphics cards, or other hardware.

Installing Service Pack 3.0 (or Higher)

Detailed instructions for installing your service pack are provided on the installation disk. Please follow those instructions to install the service pack.

Installing Microsoft Office Components

You can install Microsoft Office from the Microsoft Office CD. The installation program provides detailed instructions. Please follow those instructions to install the Microsoft Exchange messaging client, ODBC connection client, and any other components of Microsoft Office that you choose to include.

Disk-Volume Configuration

Take the following steps to configure disk volumes for your datamart:

- Step 1:** The disk volumes for your database server use NTFS format. Some computer models running Windows NT Server configure disks with FAT format by default. To convert disks from FAT to NTFS, use the Disk Administrator. From the Start Menu, choose Programs, then Administrative Tools, and then Disk Administrator.
- Step 2:** Refer to the disk-volume size figures that you calculated using the formulas in "Hardware Requirements for Windows NT Server" on page 12 for size and RAID-level information.
- Step 3:** If you are using a hardware RAID controller or RAID software, follow the manufacturer's instructions for configuring your disk volumes. Otherwise, use the Disk Administrator to configure your disk volumes.
- Step 4:** In a command-window prompt, enter the following command to enable monitoring of disk I/O performance with the PerfMonitor utility:

```
diskperf -y
```

Network Connectivity

Epiphany software requires the TCP/IP network protocol. Please contact your network administrator for specific instructions regarding the installation and configuration of devices, drivers, and domains for this protocol.

When you have verified that you have network connectivity, you can proceed to the next stage of preparations. Refer to "Installing and Configuring SQL Server" on page 23 for further instructions.

Installing and Configuring a Solaris Host

This section provides guidelines for installing and configuring a computer running Solaris 2.6 as an EpiCenter host. This section discusses a number of topics involved in installing and configuring Solaris for use with Oracle8. For additional information, refer to the *Oracle8 Installation Guide for SunSPARC Solaris* and *Oracle Support Bulletin 104511.69*.

For information about a Windows NT Server host, see “Installing and Configuring a Windows NT Server Host” on page 12.

Hardware Requirements for Solaris

Please ensure that your computer meets the following minimum requirements:

- At least one CPU (two or more are preferred) with a speed of 200 megahertz or higher, capable of supporting Solaris 2.6
- At least 128 megabytes of RAM

For best performance, Epiphany suggests equipping your Solaris host with its full complement of RAM.

- Swap space of at least three times the size of RAM.
- A Solaris-supported CD-ROM drive that uses High Sierra or ISO 9660 format with the Rockridge extension
- The computer must be connected to a network and have the TCP/IP interface installed.
- An adequate amount of disk space:

The size of your data warehouse depends on the amount of data that you extract from your source systems. If you already have specifications for your datamart, you can use that information to estimate the size of your datamart data, taking into account the following considerations:

- The datamart includes two copies of the database and aggregates that are between three and ten times the base-table size, with additional space included for staging tables, indexes, keys, and 100 megabytes of Epiphany metadata.

- Epiphany recommends that you provide space for temporary tables that is at least twice the size of your largest fact table, and space for rollback segments that 10 percent of the size of your datamart.
- Epiphany recommends that you use separate devices for redo logs and archived log files.

Based on these considerations, Epiphany suggests that you configure a single RAID-1+0 array for your EpiCenter datamart, and that you use the following formula to calculate its size:

Use the following formula to determine the size of this volume:

```
data_size =      2 * (aggs * fact_rows * (100 + fact_row_width))
               + 2 * (dimension_rows * (20 + dimension_row_width)) + 400MB
               + (0.1 * mart_vol_size) + 10MB
               + tmp * fact_rows * (100 + fact_row_width)

rollback_seg_size =      mart_size / 5

mart_size = data_size + rollback_seg_size
```

Replace *aggs* with a value between 3 and 10, depending on the degree to which your application requires aggregate data. Use a higher value for applications that require extensive use of aggregate data. Replace *tmp* with a value between 3 and 10, depending on your application. Use a value of 3 if you plan to install the Magnitude option.

Epiphany also suggests that you configure a separate volume for redo logs, and an additional volume for log archives. These volumes need not be RAID volumes.

The Magnitude option requires two additional RAID-0 volumes for external sort space. Use the following formulas for these volumes:

```
sort_working_vol = fact_rows * (100 + fact_row_size)

sort_temp_vol = 10 * fact_rows * (100 + fact_row_size)
```

Installing and Configuring a Solaris Host

Solaris Installation

Follow the directions provided with your Solaris software to install the operating system and current patch from Sun Microsystems.

Configuration of Solaris Resources

You must configure the following Solaris resources to support an Oracle database server for your EpiCenter datamart:

- Semaphores, shared-memory segments, and the paging threshold in the **/etc/system** file
- User **oracle** and group **dba**
- Automatic start-up and shut-down for Oracle
- Disk volumes
- Network Connectivity

The sections that follow describe these activities.

Updating the /etc/system File

As **root**, you must edit the **/etc/system** file to configure the following system resources for Oracle:

- semaphores
- shared-memory resources
- swapping threshold

Increasing values for these parameters allocates more resources which effectively reduces the amount of physical memory that is available to processes. Add or modify the following lines in **/etc/system** to configure these resources:

```
set semsys:seminfo_semmni=70
set semsys:seminfo_semmns=100
set semsys:seminfo_semmns=200

set shmsys:shminfo_shmseg=10
set shmsys:shminfo_shmmni=100
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmax=max_shared_size

set lotsfree=pages
```

Replace *max_shared_size* with a value that is between 80 percent and 90 percent of the size of physical memory in bytes. The value for **shmmax** indicates a total allowable size for shared memory; it does not allocate any memory resources.

Replace *pages* with the minimum number of pages that your computer requires in order to prevent unnecessary swapping. Epiphany recommends a value between 2 percent and 6 percent of the total number of pages in RAM. Use the following formula calculate the number of 8-kilobyte pages in RAM:

$$total_pages = megs * 128$$

Replace *megs* with the number of megabytes of RAM that are installed on your computer.

Use the **reboot** command to reboot your Solaris host so that these semaphore and shared-memory settings can take effect.

Creating User and Group IDs

When your operating system has finished rebooting, log in as **root**. Use **admintool** or the **useradd** utility to create the **oracle** and **epichnl** user IDs and the **dba** group. Set the default group ID for both **oracle** and **epichnl** to **dba**. Propagate this account and group information to the Network Information Service (NIS).

Installing and Configuring a Solaris Host

Modifying the Shell Initialization Files

Take the following steps to modify the shell initialization files for the **oracle** and **epichnl** user IDs.

Edit the **.profile** file in the **oracle** home directory to add the following lines. You can either type them in or copy and paste them from the Epiphany installation CD. After you place the installation CD in the CD-ROM drive, you can use a text editor to display the **/cdrom/unix/profile.sh** file.

```
umask 022
ORACLE_BASE=/opt/oracle # or some other pathname
ORACLE_SID=epi
ORACLE_HOME=$ORACLE_BASE/product/version
ORACLE_TERM=termtype
PATH=.:$ORACLE_HOME/bin:$ORACLE_BASE/local:/bin:/usr/bin:\
/usr/ccs/bin:/usr/openwin/bin:/usr/5bin:/usr/ucb
DISPLAY=remote_workstation:0.0 # remote installation only
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ORACLE_HOME/lib
TNS_ADMIN=$ORACLE_HOME/network/admin
ORA_NLS33=$ORACLE_HOME/ocommon/nls/admin/data
export ORACLE_BASE ORACLE_SID ORACLE_HOME ORACLE_TERM PATH \
DISPLAY LD_LIBRARY_PATH TNS_ADMIN ORA_NLS33
ORAENV_ASK=no
```

If you use the C shell, add the following lines the **.cshrc** file as well, which you can copy from the **/cdrom/unix/cshrc.csh** file of the installation CD.

```
umask 022
setenv ORACLE_BASE /opt/oracle # match pathname in .profile
setenv ORACLE_SID epi
setenv ORACLE_HOME $ORACLE_BASE/product/version
setenv ORACLE_TERM termtype
setenv PATH .:$ORACLE_HOME/bin:$ORACLE_BASE/local:/bin:\
/usr/bin:/usr/ccs/bin:/usr/openwin/bin:/usr/5bin:/usr/ucb
setenv DISPLAY remote_workstation:0.0 # remote installation only
setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:$ORACLE_HOME/lib
setenv TNS_ADMIN $ORACLE_HOME/network/admin
setenv ORA_NLS33 $ORACLE_HOME/ocommon/nls/admin/data
set ORAENV_ASK=no
```

In each file, replace *termttype* with the appropriate value for your terminal or terminal emulator. Replace *remote_workstation* with the name of the workstation or monitor from which you intend to perform the Oracle installation.

- Step 5:** Log in as user **epichnl** and copy the **.profile** and **.cshrc** files that you just edited from the **oracle** home directory to the home directory for **epichnl**.
- Step 6:** Enable remote execution access for user **epichnl** by adding the following line to the **.rhosts** file in the **epichnl** home directory:

```
application_host      epichnl
```

Replace *application_host* with the hostname of the Windows NT Server host computer on which you intend to install your Epiphany application. Entries in the **.rhosts** file are case sensitive, so be sure that you enter the hostname exactly as it appears in the NT domain.

Setting Up Automatic Start-up and Shut-down for Oracle

- Step 1:** As **root**, set up the automatic start-up and shut-down script for Oracle, and another for the Oracle system identifier by entering the following command lines into the **/etc/init.d/dbora** file:

```
ORACLE_HOME=oracle_home_path
ORACLE_OWNER=oracle
if [ ! -f $ORACLE_HOME/bin/dbstart -o ! -d $ORACLE_HOME ]
then
    echo "oracle startup: cannot start"
    exit
fi
case "$1" in
'start')
    su - $ORACLE_OWNER -c $ORACLE_HOME/bin/dbstart &
;;
'stop')
    su - $ORACLE_OWNER -c $ORACLE_HOME/bin/dbshut &
;;
esac
su - $ORACLE_OWNER -c "lsnrctl start"
```

Installing and Configuring a Solaris Host

Replace *oracle_home_path* with the value of ORACLE_HOME as it appears in the **.profile** file you created in the previous section.

Step 2: Create links to the **dbora** file in the system-initialization run-level directories:

```
ln -s /etc/init.d/dbora /etc/rc0.d/K10dbora
ln -s /etc/init.d/dbora /etc/rc2.d/S99dbora
```

Step 3: Create a symbolic link for the **listener.ora** file:

```
ln -s /var/opt/oracle/listener.ora /etc/listener.ora
```

Disk-Volume Configuration

As **root**, take the following steps to configure disk volumes for your datamart:

Step 1: Refer to the disk-volume size figures that you calculated using the formulas in “Hardware Requirements for Solaris” on page 16 for size and RAID-level information.

Step 2: If you are using a hardware RAID controller or RAID software, follow the manufacturer’s instructions for configuring your disk volumes. Otherwise, use the standard Solaris utilities for configuring disk partitions.

Step 3: Create three mount points on these disk volumes for the datafiles from which your Oracle tablespaces can be constructed.

Network Connectivity

The following steps enable network access for the Oracle8 database server:

Step 1: Add the following entry to the **/etc/services** file for your Oracle database server:

```
listener      1521/tcp      #TNS Listener
```

Step 2: Log in as a user other than **root**. Then use the **ftp** command to verify that your Solaris host is connected to the network and can transfer data.

When you have verified that you have network connectivity, you can proceed to the next stage of preparations. See “Installing and Configuring Oracle” on page 29 for further instructions.

INSTALLING AND CONFIGURING YOUR DATABASE SERVER

This section provides guidelines and recommendations for:

- Installing and configuring the database server for your EpiCenter datamart
- Assigning data stores for your EpiCenter data
- Creating databases for your EpiCenter datamart and metadata

Installing and Configuring SQL Server

This section provides guidelines and recommendations for installing and configuring SQL Server for use with an EpiCenter datamart.

Note: For enhanced performance, Epiphany recommends that you install the Enterprise Edition of Version 7.0. This Edition can be installed only on hosts that are running the Enterprise Edition of Windows NT Server.

Installing SQL Server

Note: Before you install SQL Server 7.0, you must install Internet Explorer 4.01, Service Pack 1, from the Install Prerequisites dialog box of the SQL Server installer.

To install SQL Server, follow the directions provided on the installation CD, along with the following recommendations.

1. Use the default path for the SQL Server installation path.
2. Use the default value for Master-device creation.
3. Choose the 850 Multilingual character set in the Select Character Set dialog box.
4. Choose the default Sort Order (dictionary order, case insensitive).

Installing and Configuring SQL Server

5. Choose the default settings for Unicode collation.
6. Select TCP/IP Sockets in the Select Network Protocols dialog box.
7. Choose the automatic start-up option for both SQL Server and SQL Executive.
8. When asked for the SQL Execution Log on Account, enter your password.
(You must have administrator privileges.)
9. For the TCP/IP Socket Number, use the default port number.

Configuring SQL Server

Epiphany recommends that you take the following steps to configure SQL Server:

- Step 1:** Configure selected database-server parameters.
- Step 2:** Configure SMP concurrency.
- Step 3:** Disable default serialization.
- Step 4:** Configure TCP/IP sockets as the networking interface for client applications.

The following sections describe these steps.

Configuring Database-Server Parameters

To configure database-server parameters for use with EpiCenter, log on as user **sa** and start the SQL Enterprise Manager. Then take the following actions:

- Step 1:** Select Register Server from the menu. Specify **sa** as the login ID and leave the password field blank. By default, SQL Server sets up the **sa** account with a null password. You should change this password as soon as it is convenient for you to do so.
- Step 2:** Enter the machine name in the server box and choose Server, then Register Server.

Step 3: In the Server Manager window, right-click the icon for the server you just installed in the Microsoft SQL Servers tree, then choose the Configure option. Select the Configuration tab in the Server Configuration/Options dialog box. Change the following values as indicated:

- Locks: Set the value to 20,000.
- Memory: Assign as much memory as your system configuration allows. For example, 100,000 2-kilobyte blocks equals 200 megabytes of memory. Consult the *SQL Server Administration Guide* for more information.

The value that is displayed is a theoretical maximum, *not* the maximum level that the machine will handle. If you set the memory value higher than a value your machine can run, SQL Server will not restart.

- Open objects: Set the value to 15,000.
- Procedure Cache: Set the value to between 5 and 15, depending on your application

Configuring SMP Concurrency Parameter for SQL Server 7.0

SMP Concurrency controls the number of threads that SQL Server releases to the operating system. The effect of this action is to limit the number of CPUs that SQL Server uses. On a uniprocessor computer, the optimal value is 1. On a symmetric multiprocessor (SMP) computer, the value you choose depends on whether or not the host is dedicated to SQL Server. If the host is dedicated to database-server operations, you can set SMP Concurrency to -1, which automatically allocates all available CPUs to SQL Server.

If the server is not dedicated, then you must allow applications other than SQL Server to have adequate CPU resources. In particular, if you install Epiphany applications on the same host as your datamart, you must allow enough resources for those applications to provide adequate response time to users. Epiphany suggests that you set SMP concurrency to one or two less than the number of CPUs for computers that host both your datamart and your Epiphany application.

Installing and Configuring SQL Server

The default value for SMP concurrency is 0 on an SMP machine, in which case SQL Server automatically reserves one less than the number of CPUs on board. On a uniprocessor machine, SMP concurrency is set to 1. If you choose Dedicated SMP Support, SMP concurrency is set to -1.

To set SMP concurrency, follow these steps:

- Step 1:** From the Enterprise Manager, select the name of your server and right-click Configure.
- Step 2:** In the Configuration tab, set the Show Advanced Options value to 1.
- Step 3:** When the advanced parameters appear in the table of configuration parameters, set the appropriate value of the SMP Concurrency parameter for your host.

Disabling Default Serialization

The Epiphany Application Server makes several SELECT INTO type queries to calculate results. In the default SQL SERVER configuration, these types of queries attempt to serialize themselves by blocking on a particular table. To avoid this problem, you should set the -T5302 flag. Follow these steps:

- Step 1:** In the Enterprise Manager window, right-click the Server.
- Step 2:** Select Configure.
- Step 3:** Click Parameters.
- Step 4:** Add -T5302.

For more information about this flag, consult the Microsoft knowledge-base-article number Q153441: "*SELECT INTO Locking Behavior*."

To obtain access the Microsoft knowledge base, visit the Microsoft support site at <http://www.microsoft.com/support> and follow the directions to view on-line documentation.

Configuring TCP/IP Sockets

Epiphany applications use the TCP/IP protocol to communicate with SQL Server. Take the following steps to configure set up TCP/IP sockets as the networking interface for SQL Server:

- Step 1:** From the Start Menu, choose Programs, then Microsoft SQL Server, and then SQL Client Configuration Utility.
- Step 2:** In the Net Library tab, select TCP/IP sockets from the Default Network pop-up menu, then click Done.

Assigning Data Stores in SQL Server

Use the SQL Server Manager to set up new database devices and new databases in your EpiCenter datamart. You need to set up the following database devices:

1. A device for your datamart data
2. A log device for your datamart data
3. A device for Epiphany metadata
4. A log device for metadata
5. A temporary-database expansion device
6. A log device for the temporary database

Note: Epiphany recommends that you always use different devices for data and logs. The naming scheme you choose for your devices should distinguish between related data and log devices. For example:

- EMRT1.DAT and EMRTLOG1.DAT for datamart data and logs
- EMTA1.DAT and EMTALOG1.DAT for metadata and logs
- ETMP1.DAT and ETMPLOG1.DAT for the expanded temporary database

Take the following steps to set up each database device:

- Step 1:** Choose your server the SQL Server Manager window, then right-click on Database Devices.
- Step 2:** Choose New Device from the pop-up menu.

Installing and Configuring SQL Server

Step 3: Enter the name for your device.

Step 4: In the Location pop-up menu, select the disk volume that you created for the device you are creating (datamart, datamart log, metadata, and so on) and assign the amount of space on that disk volume that you intend to allocate to the device. (Refer to “Hardware Requirements for Windows NT Server” on page 12 for information about the size calculations for disk volumes and data stores.)

Step 5: Click Create Now to create the device.

Creating Databases in SQL Server

After creating the new devices, you need to set up the databases for your datamart:

Step 1: Right-click the Database directory in the SQL Server Manager window.

Step 2: Select New Database from the pop-up menu. The New Database dialog box) is displayed.

Step 3: Enter the name of a database: either *epimeta* or *epimart*.

Step 4: Select the device you just created for your datamart.

Step 5: Select the log device for this database.

Step 6: Click Create Now to create the database.

Create the second database by following the same steps.

Configuring the Temporary Database

The default size of the temporary database, **tempdb**, is 2 megabytes. The default location for **tempdb** is in the master device for SQL Server. Epiphany recommends that you expand this database and place it in a separate device. This process involves placing **tempdb** in RAM, then moving it to the intended device. For information on moving and expanding the temporary database, please refer to the SQL Server documentation and Microsoft knowledge-base-article number Q187824: “*How to Move TEMPDB to a Different Device.*”

This completes the configuration instructions for SQL Server and the preparations required for a datamart host running Windows NT Server. For information on preparing your Epiphany application host, refer to “Preparing Your Application Host” on page 35.

Installing and Configuring Oracle

This section provides guidelines and recommendations for installing and configuring Oracle8 for use with an EpiCenter datamart.

Installing Oracle

Follow the directions provided in the *Oracle8 Installation Guide* to install Oracle. The following steps summarize this process:

Step 1: Log in as user **oracle**, **cd** to **/cdrom/oracle805/orainst**, and enter one of the following commands:

```
./orainst /c          # for character-based installation
./orainst /m          # for Motif-based installation
```

Step 2: Select Default Install, then Install, Upgrade, or Deinstall Software, and then Install New Product with Database Objects.

Step 3: Confirm the pathnames of the ORACLE_BASE and ORACLE_HOME directories.

Step 4: Confirm the location of the installation-log files.

Step 5: Choose Install from CD-ROM in the Install Source dialog box.

Step 6: Confirm the name of the Oracle instance you wish to create. The instance name is **epi**, the value you set previously for the ORACLE_SID environment variable.

Step 7: Install all Oracle and Solaris on-line documentation and help. Make sure that the pathname of the documentation directory is correct in the ORACLE_DOC dialog box. The default pathname is acceptable.

Installing and Configuring Oracle

Step 8: Select the following products to install:

- Oracle8 Enterprise
- SQL*Plus
- PL/SQL
- Net8
- Net8 Protocol Adapters
- Advanced Networking
- Solaris Documentation
- Oracle Solaris Installer
- On Line Text Viewer

Step 9: Choose Yes when you are asked to confirm default database start-up.

Step 10: Enter three pathnames for mount points as requested. If you have reserved separate disk volumes for Oracle data, use the pathnames to the root directory of each volume. Otherwise, use pathnames to directories within the disk volume you reserved for Oracle.

Step 11: Enter a suitable pathname for Oracle documentation when prompted. Epiphany suggest entering a pathname of the form:

`oracle_home/doc`

Replace *oracle_home* with the pathname of the home directory for your Oracle instance.

Note: At this point, the Oracle installer begins to download files. This process typically takes less than an hour, and requires periodic checking for status or error messages.

Step 12: When the installer displays the Information dialog box, choose OK in it and all subsequent dialog boxes, then exit the installer.

Configuring Oracle

Epiphany suggests that you take the following steps to configure Oracle8. For detailed configuration instructions, refer to the chapter entitled “Configuring the Oracle8 System” in the *Oracle8 Installation Guide*.

Step 1: Log in as **root** and enter one of following shell commands to clear environment variables that might adversely affect the configuration process:

```
unset SRCHOME TMPDIR TWOTASK      # Bourne or Korn shell
unsetenv SRCHOME TMPDIR TWOTASK   # C shell
```

Step 2: Review the **root.sh** script, and if it is correct, run it. If not, you can update this script and then run it without having to rerun the installer:

```
cd $ORACLE_HOME/orainst
sh ./root.sh
```

Answer Y to the following prompt if it appears:

```
ORACLE_HOME does not match the home directory for Oracle.
OK to continue? [N]:
```

Step 3: Ensure that the Oracle instance is running:

```
/bin/ps -ef | grep pmon
```

If the instance is running, the **ps** command displays a process listing that includes the SID for your Oracle instance. If your instance is not running, enter the following commands to start it:

```
svrmgrl # starts the server manager
connect internal
startup
disconnect
exit
```

Installing and Configuring Oracle

Step 4: Check to see that the Oracle listener process is running by entering the following command:

```
lsnrctl status
```

If this process is not running, lsnrctl displays a number of “unable to connect” and “no listener” messages, among others. To start the listener process, enter the following command:

```
lsnrctl start
```

Step 5: Verify that there is an entry for your EpiCenter database in the `/var/opt/oracle/oratab` file. The entry should take the form:

```
SID:ORACLE_HOME: Y
```

Replace `SID` with the instance ID of your Oracle instance. Replace `ORACLE_HOME` with the pathname listed in the `ORACLE_HOME` environment variable.

Step 6: Make sure that the Oracle command file has correct permissions, as follows:

```
cd $ORACLE_HOME/bin
chmod 4751 oracle
ls -lg oracle
```

The permissions should be:

```
-rwsr-s--x  oracle  dba  oracle
```

Step 7: Log in as **oracle** to perform the remaining configuration tasks.

Step 8: Download the Epiphany configuration scripts for Oracle, as follows:

- a) Insert the Epiphany installation disk in the CD-ROM drive on your Solaris host.
- b) Enter the following commands:

```
cd $ORACLE_HOME/rdbms/admin
cp /cdrom/unix/*.{sh ,sql,ora} .
```

Replace `SID` with the SID for your Oracle instance.

Step 9: Update the `$ORACLE_HOME/dbs/initSID.ora` file for your Oracle instance by creating a link to the `epi.ora` sample file, which you downloaded in the previous step:

```
cd $ORACLE_HOME/dbs
mv initSID.ora initSID.orig
ln -s ../admin/rdbms/admin/epi.ora initSID.ora
```

Step 10: Edit your `initSID.ora` file to provide values that are consistent with the size of your application. Please refer to the *Oracle8 Reference* for detailed information about specific initialization parameters and values.

Note: Epiphany applications typically perform large decision-support queries, as opposed to the large numbers of concurrent-but-brief queries of a typical on-line-transaction-processing (OLTP) application. Epiphany recommends that you configure Oracle using initialization parameters and values that are geared toward high TPC-D (Transaction Processing Performance Council benchmark D) performance. The sample file provides suggested values only, which Epiphany suggests that you edit to suit your application, and that you adjust over time as you learn more about performance in your specific circumstances.

Creating Control Files

Epiphany provides a sample SQL script, called `epi_rbctl.sql`, that you can edit and run to create control files and perform other initialization tasks. As you edit this script, please be aware that quoted strings and pathnames are case sensitive in Oracle SQL. By convention, Epiphany tablespaces and datafiles use uppercase names. After you have edited the script, enter the following commands to run it using `svrmgrl`:

```
svrmgrl # start svrmgrl
connect internal;
@epi_rbctl.sql
exit
```

Note: The `epi_rbctl.sql` script calls several system-catalog-creation scripts that produce copious status messages and a number of “drop object” warnings that you can safely ignore.

Assigning Data Stores in Oracle

Epiphany provides a UNIX shell script, called **make_tablespaces.sh**, that you can use as an aid in configuring tablespaces for your datamart. Based on your input, this script produces an SQL script that contains the appropriate CREATE TABLESPACE commands for your datamart. If you have not already done so, follow the instruction in Step 8 of the Configuring Oracle section to download this script.

Use the **make_tablespaces.sh** shell script to specify the number of datafiles for the following tablespaces:

- Large fact tables
- Indexes on large tables
- Dimension tables
- Indexes on dimension tables
- Metadata tables
- Transient tables
- Other application-specific tables

Take the following steps to use this script:

Step 1: Run the **make_tablespaces.sh** script by entering the command:

```
$ORACLE_HOME/rdbms/admin/make_tablespaces.sh
```

Step 2: Review the **make_tablespaces.sql** script to ensure that the sizes and pathnames for datafiles are correct.

Step 3: Enter the following commands to execute the SQL script:

```
sqlplus oracle  
-- enter password  
@maketablespace.sql
```

Note: This SQL script creates tablespaces with specific names that EpiManager and EpiChannel recognize by default. If you use alternate names for tablespaces, you must define values for the macros that EpiChannel uses to locate those tablespaces. Refer to the *Epiphany System Guide* for details on Epiphany macros.

Creating Users for EpiMart and EpiMeta Tables

Epiphany provides a sample SQL script, called **epi_user.sql**, that you can use to create standard users (owners) for EpiCenter datamart and metadata tables. Edit this sample script, which you downloaded to **\$ORACLE_HOME/bin** in Step 8 of the previous section, and then use **sqlplus** to run it. Please be aware that user names in Oracle must be all uppercase.

```
sqlplus internal # start sqlplus
@epi_user.sql
exit
```

This completes the configuration instructions for Oracle and the preparations required for the datamart host. For information on preparing your Epiphany application host, please proceed to the next section.

PREPARING YOUR APPLICATION HOST

Epiphany applications require a minimum of 64 megabytes of RAM on your application host. You must install the following software packages on your application host before you install Epiphany software:

- Windows NT Server 4.0
- Microsoft Internet Information Server (IIS), Version 4.0
- Selected components of Microsoft Office

If your application host is not the same computer as your datamart host you must also:

- Install the appropriate client software for the database server on which your datamart resides.
- Configure connectivity for your client software.

The following sections describe these actions.

Installing Windows NT Server 4.0 on Your Application Host

If you plan to install your Epiphany application on the same host as your datamart, you do not need to install Windows NT Server again. Otherwise, please see, “Windows NT Server 4.0 Installation” on page 13, and “Network Connectivity” on page 15 for instructions on installing this operating system and configuring it.

Installing Internet Information Server, Version 4.0

Epiphany requires that you install Microsoft Internet Information Server (IIS) 4.0. IIS 4.0 is distributed with Windows NT Server, Options Pack 4.0. Detailed instructions for installing components of this options pack are provided on the installation disk. You can install additional components of the options pack on your application host if you choose.

Follow these recommendations when setting up IIS:

- Use the default values for the Options dialog box.
- You may specify different values in the Database Publishing Directory dialog box. To use the security features of IIS, be sure that the **wwwroot** directory resides on an NTFS file system.
- Select the Microsoft SQL Server driver to install.
- Set the Time Zone.
- In the Windows NT Server Services dialog, set the following services to Manual start-up: Certificate Authority and Content Index.

After installing the package, please reboot Windows NT Server.

For further information about IIS 4.0, please refer to the Microsoft support site, <http://www.microsoft.com/support>.

Installing Microsoft Office Components

You can install Microsoft Office from the Microsoft Office CD. The installation program provides detailed instructions. Please follow those instructions to install the Microsoft Exchange mail client, ODBC connection client, and any other components of Microsoft Office that you choose to include.

Installing and Configuring Connectivity Packages for Your Database Client Software

If you plan to install Epiphany software on the same computer as your datamart, you do not need to install or configure additional client software. You can instead turn to Chapter 2, “Installing Epiphany Software.” If you plan to install your Epiphany application on a separate computer, please proceed with the sections that follow.

SQL Utilities

If you are using SQL Server for your datamart, you must install the SQL Utilities, option on the SQL Server installation CD, which provides appropriate instructions. After you have installed SQL Utilities, follow the steps in “Configuring TCP/IP Sockets” on page 27 to enable TCP/IP connectivity.

Oracle8 Client

If your datamart resides on an Oracle database server, you must install the Oracle8 Client package for Windows NT Server. Please observe the following recommendations:

- In the Select Installation Options dialog, choose Oracle8 Client.
- In the Select Oracle8 Client Configuration dialog, choose Database Administrator.
- Epiphany suggests that you install the Oracle documentation set on your hard drive.

Installing and Configuring Connectivity Packages for Your Database Client Software

The Oracle installer installs Net8 as part of the client-utilities package. You use Net8 to establish connectivity with your datamart. To establish a connection to the Oracle instance on which the datamart resides, you must provide Net8 with the following information about that instance. You can use the Net8 Easy Config utility that comes with Oracle8 Client, or you can update the **tnsnames.ora** file directly. Either way, you must supply the following information:

- The protocol, in this case TCP/IP
- The hostname or IP address of the datamart host
- The port number of the TNS listener service on the datamart host
- The Oracle instance name (SID)

The **tnsnames.ora** file is located in the **Orant\Net80\Admin** folder in your Oracle installation directory. A typical entry for an Epiphany datamart takes the following form:

```
EPI=
  (DESCRIPTION=
    (ADDRESS=
      (PROTOCOL = TCP)
      (HOST = epihost)
      (Port = 1521)
    )
    (CONNECT_DATA = (SID = EPI)
  )
)
```

For additional details on configuring Net8, refer to the *Net8 Getting Started* manual.

This completes the tasks that you must perform to prepare your Epiphany application host. Please turn to Chapter 2 for instructions on installing Epiphany software.

INSTALLING EPIPHANY SOFTWARE

This chapter describes the procedures for installing and configuring Epiphany application software. The standard software installation includes:

- the EpiManager™ administrative utility for configuring and managing your datamart and applications.
- the AppServer™ application server, which coordinates user requests for data and reports, returns results in HTML format, and supports navigation between applications.
- the EpiChannel™ extraction utility for downloading data from source systems into your EpiCenter datamart.
- other Epiphany components and utilities.
- drivers and other third-party software components on which Epiphany software depends.

Note: Epiphany application software does not have to be installed on the same computer on which your EpiCenter datamart resides.

The computer from which you run the Epiphany Application Server (AppServer™) must provide:

- Windows NT Server 4.0 set to run in 256-colors mode or higher
- 64 megabytes of RAM for use by Epiphany software

Installing Epiphany Software

- **Microsoft Internet Information Server (IIS), Version 4.0**
If IIS is not already installed on your application host, you must install it before you attempt to install Epiphany software. Please review the guidelines for installing IIS listed in “Installing Internet Information Server, Version 4.0” on page 36.
- **Selected components of Microsoft Office**
If the Microsoft Exchange Messaging client and the ODBC connection client are not yet installed on your application host, you must install it. Please review the guidelines for installing these components described in “Installing Microsoft Office Components” on page 37.

If your EpiCenter datamart resides on a Windows NT Server host, you can use the same computer for your datamart and your Epiphany application.

The Epiphany installer offers to install several third-party system components that are required to support your Epiphany applications if they are not already present on your system. These components include:

- ODBC and JDBC drivers
- Sun JRE
- Microsoft Internet Explorer 4.0 (IE), including the Microsoft Java virtual machine

Please install IE 4.0 if the installer asks you to do so. The installer prompts you to exit if the Java virtual machine is not present and you do not install it.

After you have installed Epiphany software on the on the application host, you can install individual components of the EpiCenter Enterprise Manager (also referred to as EpiManager™) administrative utility on other computers to provide easier access for specific administrative tasks.

To install the complete set of standard Epiphany software on your application host, choose the Application Server installation option. To install an additional copy of selected Epiphany utilities on a remote host (which can run Windows 95, Windows 98, or Windows NT Server 4.0), choose the Remote Administration option.

Figure 2 on page 41 illustrates the components that you install on the Epiphany application host and any remote-administration hosts that you might add.

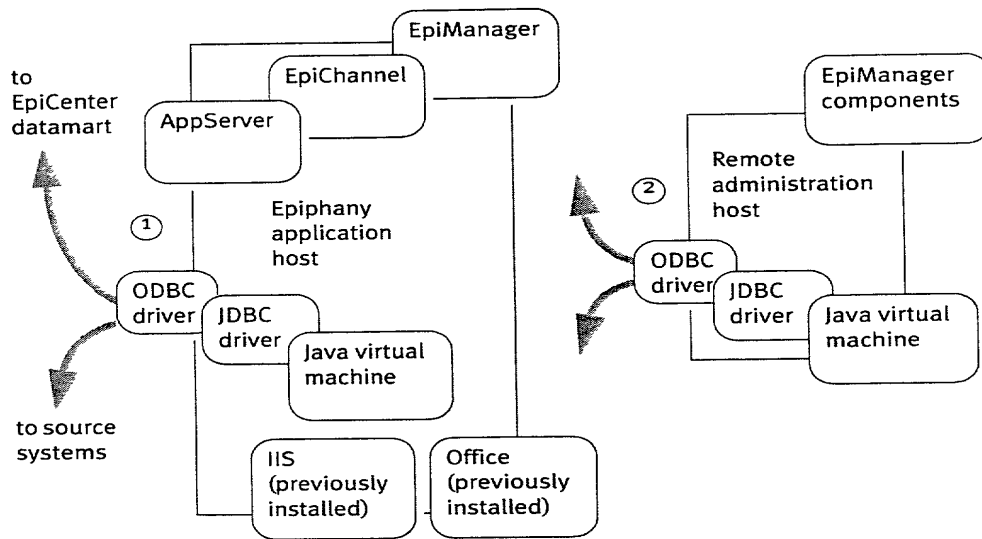


Figure 2: Epiphany Application Components

PERFORMING THE INSTALLATION

Take the following steps to install Epiphany software on your application host:

Step 1: Exit all open Windows applications.

If an AppServer instance is running, from the Start menu, go to **Settings\Control Panel\Services** and manually stop the Application Server. If you do not, the installer cannot properly download the DLLs that this release requires.

Step 2: Insert the Epiphany CD into your CD-ROM drive.

Application Server

Step 3: Double-click the **Setup.exe** icon in the CD-ROM folder.

Step 4: If the Microsoft Java virtual machine is not installed on your system, you are requested to install it. Please choose Yes in response to this prompt.

Step 5: The System Configuration screen displays your system's configuration data (for your information).

Follow the instructions below for either the Application Server or Remote Administration installation options. If you are installing Epiphany software for the first time, choose the Application Server option.

Application Server

The following steps apply to the Application Server installation option.

Step 1: Choose the Application Server option.

A Services window shows which services will be stopped before the copy operation begins.

Step 2: Enter the instance name. This is the name of the service as it will appear in the Service Control Panel.

An instance name distinguishes among multiple installations of the Epiphany software on the same machine. Typically, you will install the software once on a machine and thus have one instance. You are asked if this instance will be the default instance. If so, the **Setup** program makes this instance the default Web server destination.

Step 3: Enter the machine name on which the Application Server runs.

By default, this is the full DNS name of the server. Because of your local network configuration, you may need to use a unique name; for example, the machine name without the domain name. Check with your local system network administrator if you have any questions.

Step 4: Accept the default TCP/IP port for the Application Server if there is only one instance of the Application Server.

Each instance must have a separate TCP/IP port. Check with your local system network administrator if this is the case.

Step 5: If your datamart resides on SQL Server, enter the name of the database server and the database name. The database name is the name of the EpiMeta database. This database does not need to exist as of yet. (The Registry keys are populated based on what you enter.)

If your datamart resides on an Oracle database server, enter the service name of the Net8 service associated with your datamart.

Step 6: If your datamart is on SQL Server, enter the user name and password for the database server. The user must have system administrator (SA) privileges on the EpiCenter host.

If your datamart resides on Oracle, enter the user name for your EpiMeta user (typically **EPIMETA**) and the password for that user.

Step 7: At this point, the installer asks if the instance name you entered in Step 1 is the default instance name. If you are installing Epiphany software for the first time, or if you plan to run a single instance of AppServer on this host, click Yes.

If you click No, your Web Server will not be set up to automatically route users to the Epiphany login page. In that case, users can access the login page by entering:

`http://machine_name/scripts/instance_name/Epiphany.dll`

Step 8: Select the destination directory for the Epiphany software. The default directory is:

`C:\Program Files\Epiphany\instance_name`

Step 9: Enter the location of the run-time reports and charts that end users will access. The default location is:

`C:\Program Files\Epiphany\instance_name\Charts`

Step 10: Create a new folder as requested.

Remote Administration

Step 11: Select the components to be installed:

- Administrative Tools

You have the option of selecting individual components within the Administrative Tools option. If you are installing Epiphany software for the first time, choose the components that have been selected by default.

- Documentation

Epiphany provides technical manuals on line in PDF format with keyword-search indexes enabled. To use the PDF full-text index, you must have version 3.0 or higher of Acrobat Exchange or Acrobat Reader. You can download Acrobat Reader 3.0 from the Adobe Web site:

<http://www.adobe.com>.

Step 12: Select the default program-icon location.

Step 13: If this is an upgrade from Epiphany Release 3.2 or 3.3, and your datamart is already populated with data, refer to Chapter 3, "Migrating From Release 3.2 or 3.3" for further instructions. Otherwise, you can go to the Service Control Panel and start the service.

This completes the Application Server installation option.

Remote Administration

Step 1: Select the destination directory for the Epiphany software. The default is **C:\Program Files\Epiphany\instance_name**.

Step 2: Enter the location of the run-time reports and charts that end users will access. **C:\Program Files\Epiphany\instance_name\Charts** is the default. Create a new folder as requested.

Step 3: Select the components to be installed:

- **Administrative Tools**

You have the option of selecting components of the following Administrative Tools: EpiCenter Manager, Web Builder, and Security Manager. Although EpiCenter Manager includes Web Builder and Security Manager, these are available as separate programs. A person who does not need to have access to all of the features of EpiCenter Manager can use one of these components. The ticksheet-configuration features of EpiCenter Manager are available via Web Builder. Access rights and system permissions are available via Security Manager.

- **Documentation**

Step 4: Select the default program-icon location.

Step 5: If you are performing an upgrade from Epiphany Release 3.2 or 3.3 and your datamart is already populated with data, please refer to Chapter 3, “Migrating From Release 3.2 or 3.3” for further instructions. Otherwise, you can go to the Service Control Panel and start the service.

This completes the Remote Administration software-installation option.

CONFIGURING EPIPHANY SOFTWARE

After you have installed Epiphany software, you must configure the Epiphany Web-server proxy. In addition to configuring the proxy, Epiphany also recommends that you enable performance monitoring for data-extraction operations.

Configuring the Epiphany Web-server Proxy

The Epiphany Web-server proxy interacts with IIS 4.0. For the Epiphany Application Suite to function properly, set the IIS configuration values according to the instructions in this section.

Step 1: Open the IIS 4.0 Service Manager from the Start menu by choosing Programs, then Microsoft Internet Server, then Internet Service Manager.

Setting Up NT Performance Monitoring for Extraction

Step 2: Right-click your server icon and select Properties from the pop-up menu. The Properties dialog box is displayed in the Master Properties panel. Select WWW Services as the Master Properties. Click Edit. In the WWW Service Master Properties dialog box, select Directory Security. In the Anonymous Access and Authentication Control Panel, click Edit. The Authentication Methods dialog box is displayed. Select Allow Anonymous Access. Click Edit.

Step 3: Configure Anonymous Login with file access permissions for reading and writing to all Epiphany files. Enter the following attributes for the Anonymous user account:

- Username
A username that has file access permission for all of the Epiphany installed files and directories. This username also needs permission to read the Epiphany entries in the Registry.
Make sure that the directories that contain the **Epiphany.dll** and the **makechart.dll** files have execute permissions.
- Password
The password for this username.

Setting Up NT Performance Monitoring for Extraction

The **extract.exe** program is instrumented to use the standard NT Performance Monitoring facility, but you must take the following steps to enable the display of Epiphany data extraction processes in the NT Performance Monitor:

Step 1: Your **Epiphany\instance_name\win32** installation directory must contain these items: **EpiPerfMon.dll**, **EpiPerfMon.ini**, and **EpiPerfMon.reg**.

EpiPerfMon.reg has values specified for the following Registry key:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Epi\Performance

Edit **EpiPerfMon.reg** to change the *instance_name* to the correct value for the Library path, save the file and exit.

Step 2: Run **EpiPerfMon.reg**. To check the results, open the Registry and navigate to the key specified in Step 2. Ensure that the path supplied for Library is correct. If it is not, the Epiphany Channels object entry does not appear in the object list for the Performance Monitor.

```
unlodctr Epi
lodctr EpiPerfmon.ini
```

You can now start the NT Performance Monitor and monitor the progress of data-extraction jobs performed by EpiChannel.

When you install the Epiphany software, the first screen displays your system configuration.

- Run the Epiphany tools, such as EpiManager, to configure your EpiCenter and set up EpiChannel extraction jobs. You will also use EpiCenter to construct the ticksheets that enable users to query the data in your data warehouse.

Verifying Your Installation

The *Epiphany System Guide* gives instruction on how to use EpiManager to administer your Epiphany application. Please read Chapters 1 and 2 of that guide for background information before using EpiManager.

- Enter your e-mail password to receive notification of Epiphany system status. You will use the Configuration dialog box in EpiManager to set this up. Instructions for doing so appear in Chapter 3 of that guide.
- Run the Epiphany extraction program (EpiChannel) to extract data from your source systems and place them in your EpiCenter datamart. EpiChannel performs the extraction jobs that you defined in EpiCenter Manager.
- Start AppServer, the Epiphany Application Server. Refer to the *Epiphany System Guide* for details.

09625518.072500

MIGRATING FROM RELEASE 3.2 OR 3.3

This chapter describes the steps you must take to migrate metadata from Epiphany Release 3.2 or 3.3 to Release 3.4.

EpiMeta is the physical database that contains Epiphany control information. The EpiMeta database is implemented as a relational database with many tables and integrity constraints. All installations of a specific Epiphany release use the same data model for EpiMeta. However, upgrades to the product can include changes and additions to that model. Release 3.4 includes such changes. The remainder of this chapter describes the steps you must take to migrate your existing 3.2 or 3.3 EpiMeta data to the new data model in 3.4.

EpiManager provides a facility for upgrading your existing installation. Rather than operating on the EpiMeta database itself, the upgrade procedure operates on a standard Epiphany export file in Microsoft Access format.

Note: Beginning with Release 3.4, the following data types are no longer supported by Epiphany:

- NUMBER(9)
- NUMBER(9,2)
- NUMBER(9,5)
- FLOAT

THE UNIVERSITY OF CHICAGO

THE UNIVERSITY OF CHICAGO

THE UNIVERSITY OF CHICAGO

THE UNIVERSITY OF CHICAGO

- THE UNIVERSITY OF CHICAGO**

THE UNIVERSITY OF CHICAGO



THE UNIVERSITY OF CHICAGO

Step 2: Build a new 3.4 EpiMeta database using the Initialize EpiCenter command in EpiManager, Release 3.4.

Step 3: Using the EpiManager, start importing the Release 3.2 or 3.3 export file. Specify a new file name for the “transformed” export file as prompted.

The import should proceed normally.

The transformation that occurs in Step 3 on page 51, modifies the previous export file such that all new metadata for Release 3.4 is included in the new Microsoft Access database. This modification is irreversible, which is why a new file name is requested. The export file is changed into a valid Release 3.4 export file, with the same format and information as would be contained in an export file produced using the Release 3.4 EpiCenter Manager export facility.

Note: The following caveats apply when you migrate from Release 3.2 only:

- The maximum length of names for fact and base dimension tables has been reduced from 25 to 20 characters. If these names in your EpiCenter Manager exceed 20 characters, reduce the number of characters in Release 3.2 before performing the Release 3.4 upgrade.
- The Saved Queries naming convention reflects the more rigid folder hierarchy in Release 3.4. (Folders must have unique names.) For this reason, when importing Release 3.2 saved queries into Release 3.4, a special folder called Release 3.2 Queries is created in the Public root folder. All Release 3.2 saved queries are placed in this folder; the names are altered slightly to ensure uniqueness. You can use the Report Gallery administration feature of EpiManager to move these folders into the appropriate Release 3.4 folder.
- Measure terms can no longer use MIN, MAX, and AVG in Release 3.4. Modify any Release 3.2 measure terms using these operators before upgrading.

Migrating From Release 3.2 or 3.3

09625518.072500

Copyright and Trademarks

Copyright © 1998-1999 by Epiphany Marketing Software, Inc. All rights reserved.

This document and the software it describes are furnished under license and may be used or copied only in accordance with such license. Except as permitted by such license, the contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, without the prior written permission of Epiphany Marketing Software, Inc.

This document contains propriety and confidential information of Epiphany Marketing Software, Inc. The contents of this document are for informational use only, and the contents are subject to change without notice. Epiphany Marketing Software, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

Unpublished rights reserved under the copyright Laws of the United States.

Clarity™, Relevance™, Momentum™, and Magnitude™ are trademarks of Epiphany Marketing Software, Inc. All other products or name brands are trademarks of their respective holders.

Printed in the USA

Restricted Rights Legend

Software and accompanying materials acquired with United States Federal Government funds or intended for use within or for any United States federal agency are provided with "Restricted Rights" as defined in DFARS 252.227-7013(c)(1(ii)) or FAR 52.227-19.

09625518 072500



E P I P H A N Y

SYSTEM GUIDE

3.4

09625518 072500



Service Pack 1, March 1999

09625518.072500

CONTENTS

INTRODUCTION	13
About Epiphany.....	13
About Release 3.4.....	14
About This Guide.....	14
How To Use This Guide.....	16
Using Full-Text Search with PDF Files.....	16
 CHAPTER 1	
BASIC CONCEPTS	19
Epiphany Database Schema.....	19
The EpiCenter.....	21
Dimension Tables.....	23
Dimension Roles.....	24
Date and Transtype Dimensions.....	24
Attributes.....	25
Fact Tables.....	25
Epiphany System Overview.....	27
The Epiphany System Is Metadata.....	29
The Uniform Treatment of Time.....	31
Uniform Transactional Data.....	32
Adaptive Architecture.....	35

CHAPTER 2	39
EPIPHANY DATABASE EXTRACTION	
Extraction Phases: An Overview	41
Load Phase	41
Data Merging	42
Aggregate Building	42
Data Stores	43
The Role of Jobs in Extraction	45
Extractors	46
SQL Statement Extraction Step	48
SQL Macros	49
Staging Tables	49
External Tables	50
Semantic Instance Extraction Step	51
Applying a Semantic Type	51
System Calls	53
Aggregate Building	55
The Aggbuilder Program	58
The Aggregate Building Process	59
MomentumBuilder	60
Verifying MomentumBuilder Extraction	62
Custom Fact Indexing	62
The UNKNOWN Dimension Row	64
Referring to the UNKNOWN Row during Extraction	65
The CountUnjoined Semantic Type	65
Normal Extraction Order	66
Running Jobs: EpiChannel	67
The EpiChannel Command Line	67
EpiChannel Registry Keys	69
Output Files	70
Extracting New Rows Only	70
How EpiChannel Identifies Data To Be Extracted	72

EpiChannel Debugging	74
Job Output	75
Error Messages	78
EpiChannel Debugging Levels	78
Setting Breakpoints	79
Trial Runs	79
EpiChannel Output	80
Log and Working Directories	80
Log Files versus Log Databases	81
Monitoring Jobs	81
Running the Performance Monitor	82
Mirroring: A and B Tables	83
SQL Limits for Facts	84

CHAPTER 3

EPICENTER MANAGER

85

Planning Your EpiCenters	86
Getting Started	87
Populating the Date Dimension	91
Working with an Existing EpiCenter	92
The EpiCenter Manager Window	93
Working with EpiCenter Manager	93
EpiCenter Manager Report	95
Setting Up an EpiCenter	97
Configuration	97
Base Dimension Tables	98
Defining Dimension Aggregates	102
Multiple Dimension Column Sets	105
Dimension Aggregate Browsing	105
Setting Up a Constellation	106
Defining Dimension Roles	107
Degenerate Dimensions	108
Defining Fact Tables	108
Defining Fact Columns	110

09625518.072500

Custom Fact Indexes	111
Fact Aggregate Browsing	113
Aggregation and Aggregate Grouping	115
The Default Aggregate Group	117
Measures	118
Defining a Measure	118
Measure Terms	120
Reverse Polish Notation	122
Ticksheet Types	124
Clarity	124
Relevance	124
Momentum	125
Configuring a Clarity or Relevance Ticksheet	126
General Tab	126
Assigning the Ticksheet to a Dataset	127
Assigning Access to Groups and Users	129
Attributes Tab	130
Setting Up a Glossary Entry	134
Filters Tab	134
Defining Filter Groups	137
Defining Filter Elements	138
Measure Mappings Tab	140
Adding Elements to Columns	141
Mapping Elements to a Measure	143
Verifying that All Elements are Mapped	143
Reports Tab	144
Copy Ticksheet Items	145
Configuring Relevance Ticksheets	146
Creating Default Relevance Ticksheets	147
Best & Worst	147
Profiling	147
Trends	148
Quarter Projections	149
Lifecycles	149

Aggregates for Relevance	150
Quarter Projections	150
Profiling	150
Influence	151
Best & Worst	151
Trends and Lifecycles	151
Relevance Influence Ticksheets	152
How Influence Works	152
Classification Trees	152
Regression Trees	153
Setting Up an Influence Ticksheet	153
Determining the Primary Dimension	154
Determining the Target	154
Determine the Source Attributes	154
Measure Sets	156
Setting Up Targets for the Ticksheet	158
Tips for Using Influence	159
Using Momentum Lists as Influence Attributes	159
Influence and Slowly Changing Dimensions	160
Influence and Aggregates	160
Performance Issue	160
Using Momentum	160
The Ind_Group_Joiner Fact Table	164
How to Populate the Ind_Group_Joiner Table	164
The Momentum Constellation	165
Why Use Adjacent Constellations?	166
Setting Up Momentum Adjacent Constellations	166
Additional Momentum Configuration	167
Mini-Dimensions	168
Transaction Filters	169
Clusters and Counts	170
Configuring a Momentum Ticksheet.	171
Setting Up Momentum Attributes	171
Filters	172

Setting Up Clusters and Counts on Fact Tables	173
Defining Transaction Filters	174
Default Settings on Momentum Ticksheets	176
Editing, Deleting, and Duplicating Ticksheets	176
Extraction	177
Data Stores	177
The Data Store Dialog Box	177
Modifying the Default Data Stores	180
Extractors	180
The Extractor Steps Dialog Box	182
Jobs	189
Adding Extractors and System Calls as Job Steps	192
External Tables	194
Momentum Extraction	195
Configuring E-Mail	196
Verifying that E-mail Notification Works	197
Configuring Outlook Exchange for EpiChannel E-mail Notification	
197	
Truncating Tables	198
Purging EpiMart Tables	199
Security	200
Setting Up Groups	201
Setting Up Users	205
Report Gallery	209
Folder Menu	211
Report Menu	211
Setting Permissions	212
Shared Interface	213
Setting Up Datasets	213
Generating Schema	214
Populating the Date Dimension Table	216
Exporting/Importing Metadata	217
Toggle A and B Tables	221
Web Builder	221

09625518.072500

Security Manager	222
Running the Scrutiny Debugging Tool	223

CHAPTER 4	
EPIPHANY APPLICATION SERVER	225
Starting and Stopping the Server	227
Running as a Service	227
Determining if the Application Server Is Running	228
Running as a Console Application	230
For Authentication to Work	231
Command-line Arguments	232
Refreshing the Application Server	232
The Refresh Command Line	235
Invoking RefreshApp	236
The EpiAppService Program	237
Command Syntax	238
EpiAppService Command Examples	240
The Application Server's Registry Keys	240
The Epiphany Proxy	243
Proxy Logging	243
Application Server Logging	245
Log File Location	246
Log File Naming Conventions	246
The Server Log	247
The Security Manager	248
Save and Restore Manager	249
Epiphany Applications	249
Application Server Security	250
Authentication Modules	252
Authentication Module Tips	253
Administrator Groups	256

APPENDIX A	
EPIPHANY MACROS	257
System Call Macros	257
System Call Macro Syntax	258
Epiphany SQL Macros	262
Oracle-specific SQL Macros	262
Vendor-independent Macros	263
SQL Macro Usage	264
SQL Macro Notes	265
APPENDIX B	
EPICENTER CONFIGURATION	285
General Settings	285
Transaction Types	288
Measure Units	289
Option Labels	290
Momentum Labels	291
Ticksheet Types	293
APPENDIX C	
DATE DIMENSION FIELDS	295
APPENDIX D	
PHYSICAL TYPE VALUES	299
APPENDIX E	
WRITING STAGING SQL STATEMENTS	303
Base Dimension Staging SQL Statements	303
Duplicate sskey's	306
Dimension Staging Queries with Joins	306
Constructing Base Dimension Queries with DISTINCT Fact Values	307
Fact Staging SQL Statements	307
Using External Tables as Inputs to Staging Queries	310

APPENDIX F	311
SEMANTIC TYPES	
Dimension Semantic Types	311
Slowly Changing Dimensions	311
Latest Dimension Value	313
First Dimension Value	314
Initial Dimension Load	315
Fact Semantic Types	316
Count Unjoined (Optional)	316
Custom Fact Index	317
Transactional	317
Transactional/State-like	318
Transactional/State-like/Force Close	319
Pipelined	320
Initial Load Fact	321
Reload Max Date	322
APPENDIX G	
EXPORT/IMPORT OF METADATA	323
Metadata Overview	323
Replacing Existing Metadata on Import	325
Actuals and Agg Metadata	326
Export File Format	326
APPENDIX H	
TROUBLESHOOTING	329
Registry Editor Warning	329
SQL Server Error Message	330
Cannot Connect to the Server	330
Application Server Error Messages	330
User Cannot Log In	331
Additional Action to Take If User Still Cannot Log In	333
Service Control Manager Fails to Start Application Service	333
Out of Memory	334

09625518-072500

VirtualMartDatabase Key Missing Error	334
Invalid Object DATE_O Error	334
Not a Valid Application Error	335
Internal Windows NT Error	335
EpiQuery Engine Database Connection Open Failure Exception ..	336
Charts Do Not Display	337
GIF Images Fail to Display on Web Pages	337
No Results Available for a Query	338
Result Page Error: Extraction Date Unknown	339
Web Server Message: Object Not Found	340
Browser Crashes When Retrieving Results from Application Server	341
Refresh Program Fails	342
Application Log Full Error	342
Application Server Log Security Problem	343

GLOSSARY	345
-----------------	------------

INDEX	361
--------------	------------

Parameter	Unit	Value
Mean	mm	1.2
Standard deviation	mm	0.2
Minimum	mm	0.8
Maximum	mm	1.6
Range	mm	0.8-1.6
Median	mm	1.1
Mode	mm	1.0
Skewness		0.5
Kurtosis		0.2
Correlation coefficient		0.8
Regression equation		$y = 0.8x + 0.2$
Intercept	mm	0.2
Slope		0.8
Adjusted R-squared		0.7
F-statistic		10.0
P-value		0.001
Confidence interval	mm	0.9-1.5
95% CI	mm	0.9-1.5
90% CI	mm	0.8-1.6
99% CI	mm	0.7-1.7
Mean absolute error	mm	0.1
Root mean square error	mm	0.15
Mean square error	mm ²	0.0225
Standard error of the estimate	mm	0.12
Standard error of the regression	mm	0.1
Standard error of the intercept	mm	0.05
Standard error of the slope		0.02
Standard error of the correlation coefficient		0.05
Standard error of the F-statistic		0.01
Standard error of the P-value		0.001
Standard error of the confidence interval	mm	0.1
Standard error of the 95% CI	mm	0.1
Standard error of the 90% CI	mm	0.1
Standard error of the 99% CI	mm	0.1
Standard error of the mean absolute error	mm	0.05
Standard error of the root mean square error	mm	0.05
Standard error of the mean square error	mm ²	0.005
Standard error of the standard error of the estimate	mm	0.05
Standard error of the standard error of the regression	mm	0.05
Standard error of the standard error of the intercept	mm	0.025
Standard error of the standard error of the slope		0.01
Standard error of the standard error of the correlation coefficient		0.025
Standard error of the standard error of the F-statistic		0.005
Standard error of the standard error of the P-value		0.0005
Standard error of the standard error of the confidence interval	mm	0.05
Standard error of the standard error of the 95% CI	mm	0.05
Standard error of the standard error of the 90% CI	mm	0.05
Standard error of the standard error of the 99% CI	mm	0.05
Standard error of the standard error of the mean absolute error	mm	0.025
Standard error of the standard error of the root mean square error	mm	0.025
Standard error of the standard error of the mean square error	mm ²	0.0025
Standard error of the standard error of the standard error of the estimate	mm	0.025
Standard error of the standard error of the standard error of the regression	mm	0.025
Standard error of the standard error of the standard error of the intercept	mm	0.0125
Standard error of the standard error of the standard error of the slope		0.005
Standard error of the standard error of the standard error of the correlation coefficient		0.0125
Standard error of the standard error of the standard error of the F-statistic		0.0025
Standard error of the standard error of the standard error of the P-value		0.00025
Standard error of the standard error of the standard error of the confidence interval	mm	0.025
Standard error of the standard error of the standard error of the 95% CI	mm	0.025
Standard error of the standard error of the standard error of the 90% CI	mm	0.025
Standard error of the standard error of the standard error of the 99% CI	mm	0.025
Standard error of the standard error of the standard error of the mean absolute error	mm	0.0125
Standard error of the standard error of the standard error of the root mean square error	mm	0.0125
Standard error of the standard error of the standard error of the mean square error	mm ²	0.00125
Standard error of the standard error of the standard error of the standard error of the estimate	mm	0.0125
Standard error of the standard error of the standard error of the standard error of the regression	mm	0.0125
Standard error of the standard error of the standard error of the standard error of the intercept	mm	0.00625
Standard error of the standard error of the standard error of the standard error of the slope		0.0025
Standard error of the standard error of the standard error of the standard error of the correlation coefficient		0.00625
Standard error of the standard error of the standard error of the standard error of the F-statistic		0.00125
Standard error of the standard error of the standard error of the standard error of the P-value		0.000125
Standard error of the standard error of the standard error of the standard error of the confidence interval	mm	0.0125
Standard error of the standard error of the standard error of the standard error of the 95% CI	mm	0.0125
Standard error of the standard error of the standard error of the standard error of the 90% CI	mm	0.0125
Standard error of the standard error of the standard error of the standard error of the 99% CI	mm	0.0125
Standard error of the standard error of the standard error of the standard error of the mean absolute error	mm	0.00625
Standard error of the standard error of the standard error of the standard error of the root mean square error	mm	0.00625
Standard error of the standard error of the standard error of the standard error of the mean square error	mm ²	0.000625
Standard error of the standard error of the standard error of the standard error of the standard error of the estimate	mm	0.00625
Standard error of the standard error of the standard error of the standard error of the standard error of the regression	mm	0.00625
Standard error of the standard error of the standard error of the standard error of the standard error of the intercept	mm	0.003125
Standard error of the standard error of the standard error of the standard error of the standard error of the slope		0.00125
Standard error of the standard error of the standard error of the standard error of the standard error of the correlation coefficient		0.003125
Standard error of the standard error of the standard error of the standard error of the standard error of the F-statistic		0.000625
Standard error of the standard error of the standard error of the standard error of the standard error of the P-value		0.0000625
Standard error of the standard error of the standard error of the standard error of the standard error of the confidence interval	mm	0.00625
Standard error of the standard error of the standard error of the standard error of the standard error of the 95% CI	mm	0.00625
Standard error of the standard error of the standard error of the standard error of the standard error of the 90% CI	mm	0.00625
Standard error of the standard error of the standard error of the standard error of the standard error of the 99% CI	mm	0.00625
Standard error of the standard error of the standard error of the standard error of the standard error of the mean absolute error	mm	0.003125
Standard error of the standard error of the standard error of the standard error of the standard error of the root mean square error	mm	0.003125
Standard error of the standard error of the standard error of the standard error of the standard error of the mean square error	mm ²	0.0003125
Standard error of the standard error of the standard error of the standard error of the standard error of the standard error of the estimate	mm	0.003125
Standard error		

Epiphany is the leading provider of Relationship Management (ERM) applications. Epiphany provides complete solutions for managing customer relationships by providing Web-based access to relevant information that is captured by various departments within a business organization. Epiphany applications allow you to browse through this information, gain new insights, and explore relationships and trends that might otherwise remain hidden within your various databases.

- Browse current data from throughout your organization
- Drill down to analyze specific situations in detail
- Act on that information to satisfy current customers and cultivate new ones

Clarity gives you an instant, integrated view of your company's "information capital," which can include customers, products, leads, orders, and support calls. Whatever information exists in your enterprise can be brought into Epiphany and viewed using Clarity and a standard Web browser. Simply point your mouse, make selections, and click to access the data you want in report or chart format.

Epiphany Confidential

About Release 3.4

Momentum List Manager gives you the ability to target customers quickly and easily and to generate lists of customers or individuals. Momentum's strength is that it can perform complex queries using data from multiple databases within your enterprise. A typical question that Momentum can easily answer is "Show me a list of all contacts at any customer who has purchased at least \$20,000 of products from our company in the past two quarters, has purchased at least one copy of Product A, has not purchased Product B, and is on a service contract." Construct your list by pointing and clicking in your Web browser.

ABOUT RELEASE 3.4

Epiphany Release 3.4 provides the following features and enhancements:

- Support for larger data sets than in previous versions
- Support for implementing an EpiCenter on the following database-server platforms:
 - Oracle8 on Solaris 2.6
 - SQL Server 7.0 on Windows NT Server 4.0

ABOUT THIS GUIDE

The *Epiphany System Guide* is intended for database administrators and consultants who set up and maintain an installation's datamart, as well as those who use Epiphany's Clarity, Relevance, and Momentum software to configure *ticksheets* so that users can access relevant information in the datamart. (A ticksheet is a form that allows end users to construct queries; it is called a ticksheet because users select, or tick, items on a page.)

This *Guide* consists of the following chapters and appendices:

Chapter 1, *Basic Concepts*, describes the Epiphany database schema and introduces Epiphany-related data warehousing concepts.

Chapter 2, *Epiphany Database Extraction*, describes the Epiphany database extraction process.

Chapter 3, *EpiCenter Manager*, explains how to set up your organization's data warehouse, which is called an EpiCenter. It also describes how to create and modify ticksheets. At the front end, users open a ticksheet in a Web browser and select options for the kind of data they want to display.

Chapter 4, *The Epiphany Application Server*, is an operator's guide to the component of the Epiphany ERM application suite that processes all user requests and returns query data in HTML format.

Appendix A, *Epiphany Macros*, defines the Epiphany-supplied system calls and SQL macros.

Appendix B, *EpiCenter Configuration*, describes the configuration data for a default EpiCenter.

Appendix C, *Date Dimension Fields*, defines the date dimension fields used by the Epiphany system.

Appendix D, *Physical Type Values*, defines the database type translations for the physical types used by the Epiphany system.

Appendix E, *Writing Staging SQL Statements*, explains how to write SQL statements for base dimension tables and fact tables. Instructions for using external tables as inputs to staging queries are also given.

Appendix F, *Semantic Types*, describes the dimension and fact semantic types.

Appendix G, *Export/Import of Metadata*, presents an overview of the Epiphany system's use of metadata as a basis for a discussion of how the Epiphany export/import metadata feature works.

Appendix H, *Troubleshooting*, describes the Epiphany error conditions and error messages and suggests corrective action.

The *Glossary* defines the terms used throughout this *Guide*.

HOW TO USE THIS GUIDE

A suggested approach to using this *Guide* follows:

- Step 1:** See the *Epiphany Installation Guide* for instructions on installing and configuring the software and setting up your system.
- Step 2:** Read Chapter 1, *Basic Concepts*, and Chapter 2, *Epiphany Database Extraction*, for the background material you need to set up an EpiCenter. Refer to the *Glossary* for definitions of terms.
- Step 3:** Follow the instructions in Chapter 3, *EpiCenter Manager*, to configure your EpiCenter and to construct ticksheets for your organization.
- Step 4:** Run the EpiChannel program, as explained in Chapter 2, *Epiphany Database Extraction*, to extract data from your source systems and place it into the Epiphany tables.
- Step 5:** Start the Application Server as described in Chapter 4, *Epiphany Application Server*, and verify that it is working.
- Step 6:** Refer to the appendices for more detailed information as directed throughout this *Guide*.

USING FULL-TEXT SEARCH WITH PDF FILES

The PDF file versions of the Epiphany documentation include a full-text index (**index.pdx**), which is a searchable database of all text in the Epiphany PDF documentation. A full-text search is much faster and more precise than using the standard Find command to search a document.

You can open and view a PDF file if you have Adobe Acrobat Reader installed on your system. To use this index, however, you need to have either Adobe Acrobat Exchange (the Reader plus enhancements), or Adobe Acrobat Reader with Search.

Note: You can download the free Acrobat Reader with Search from Adobe's Web site (www.adobe.com/prodindex/acrobat/readstep.html). After you register with Adobe, select *Acrobat Reader with Search* from the pop-up menu at the same step of the process as you select the language and platform.

Follow these steps to tell Acrobat which full-text index to use:

- Step 1:** Open Adobe Acrobat Exchange or Acrobat Reader with Search.
- Step 2:** From the Tools main menu command, select Search\Indexes. The Index Selection dialog box displays available indexes. The index for both the *Epiphany System Guide 3.4* and the *Epiphany Installation Guide 3.4* is **index.pdx**. If it is not selected, you need to add it.
- Step 3:** Click Add to add the index for the Epiphany PDF files. Go to **C:\Program Files\Epiphany\instance_name\Docs** and double-click the file **index.pdx**.
- Step 4:** The index is added to the search list. Click OK to close the dialog box.

Follow these steps to search for a term:

- Step 1:** From the main menu, choose Tools\Search\Query.
- Step 2:** In the Find Results Containing Text box, enter the term whose occurrences you want to find in the document or documents. The term may be one or more words, or a number, and can include wildcards (an asterisk for multiple characters and a question mark for a single character).
- Step 3:** Click Search. The documents that contain text that matches your search query are listed in rank order in the Search Results window. Double-click the document that you want to search (usually the first one in the list).

Using Full-Text Search with PDF Files

Step 4: The PDF file is displayed with the first occurrence of your search text highlighted. Use the Search Next and Search Previous buttons in the toolbar to navigate to other occurrences. (For instruction on using Acrobat Exchange or Acrobat Reader, see their online guides, which are available from the Help menu.)

Acrobat Search has tools that enable you to expand and limit your search criteria. For complete instructions on using the search feature, see the *Search Online Guide*, also available from the Help menu.

09625518.072500

CHAPTER 1

BASIC CONCEPTS

This chapter introduces the Epiphany database schema and provides an overview of the Epiphany system. Concepts integral to the Epiphany system, such as how it treats time and transactions, are also discussed.

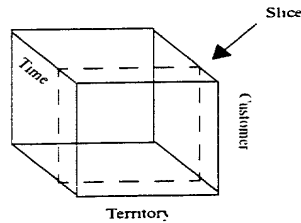
EPIPHANY DATABASE SCHEMA

The Epiphany database schema is based on the dimensional data warehouse model. A data warehouse transforms the raw data from an organization's source system databases into data accessible for query and analysis. The data conforms to the organization's business model and is consistent, reusable, and flexible (that is, the data may be re-sorted by any measure the business uses). A data warehouse also provides the tools needed to query, analyze, and publish this data.¹

The dimensional data warehouse organizes data in what may be visualized as a cube. One can figuratively slice along any dimension of the cube of data to obtain information about the intersection of the dimensions at that point. For example, if the cube has the dimensions Customer, Territory, and Time, and one selects order amount as the measure, one could slice through the cube on the Time dimension to determine the total order amount by customer or territory at a specific date.

¹ *The Data Warehouse Toolkit* by Ralph Kimball (John Wiley & Sons, Inc., 1996) is an excellent book about data warehousing.

Epiphany Database Schema



In the Epiphany system, an organization's data warehouse is known as an *EpiCenter*. (An organization may have multiple EpiCenters.) The Epiphany Application Suite consists of "front-end" Web-based applications, such as Clarity, Relevance, and Momentum, which are designed for database query and analysis. These applications allow users to query the EpiCenter by selecting dimensions and facts they want to know more about. The results of these queries (shown in reports, charts, lists, and graphs within the user's browser window) are derived from the intersection of these dimensions.

The EpiCenter represents a dimensional data warehouse with database tables organized in a *star schema* (see Figure 1, on page 21). At the center of a standard star schema is a *fact table* that contains measure data. Radiating outward from the fact table like the points of a star are multiple dimension tables. *Dimension tables* contain attribute data, such as the names of customers and territories. The fact table is connected, or joined, to each of the dimension tables, but the dimension tables are connected only to the fact table. This schema differs from that of many conventional relational databases where many tables are inter-joined.

An advantage of the Epiphany star schema is that it allows an organization's EpiCenter to be regenerated when its business model changes without the need to replace the data warehouse. The Epiphany star schema also facilitates the creation of new EpiCenters for an organization (data from existing EpiCenters can be imported).

The Epiphany star schema subsumes the standard star schema into a larger hierarchical organization (known as a *constellation*) that allows for the sharing of dimension tables by a set of similar fact tables. A constellation is a grouping mechanism for like-structured fact tables.

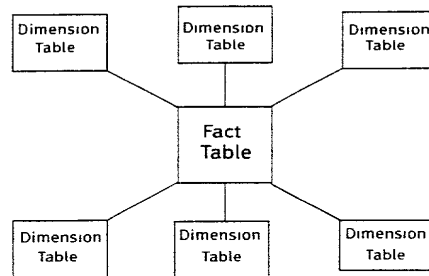


Figure 1: Standard Star Schema

As shown in the block diagram in Figure 2, on page 22, the EpiCenter is the top-level organizing principle. Each Epiphany site has at least one EpiCenter. An EpiCenter is organized into one or more constellations, and a constellation consists of a set of similar facts. Dimension tables are shared by multiple constellations within the EpiCenter.

The EpiCenter

An EpiCenter is composed of EpiMeta and EpiMart. EpiMeta refers to all of the Epiphany system's metadata tables. (*Metadata* is information about data, not data itself.) EpiMeta defines the schema for the EpiMart tables that will contain the actual extracted, organized data, such as customer, product, and order data. An EpiCenter is an EpiMeta database with its associated internal link to an EpiMart.

The EpiCenter

The EpiMart consists of fact, dimension, and staging tables. The fact and dimension tables contain actual customer data. Staging tables, which are discussed in Appendix 2, “Epiphany Database Extraction” are the first entry point of raw data from the source systems into the EpiMart—an interim stop before it reaches the EpiMart’s tables.

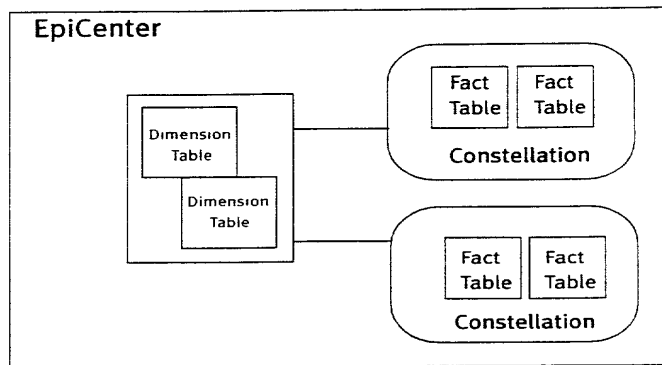


Figure 2: The Epiphany Star Schema

To facilitate the building of EpiCenters, Epiphany provides the EpiCenter Enterprise Manager, also called EpiCenter Manager. This is a Microsoft Windows application with an Explorer-like hierarchical structure (see Figure 3). The person who designs an EpiCenter (or EpiCenters) for a site uses the EpiCenter Manager’s graphical user interface to define the schema for the EpiMart tables.

09625518.072500

Dimension Tables

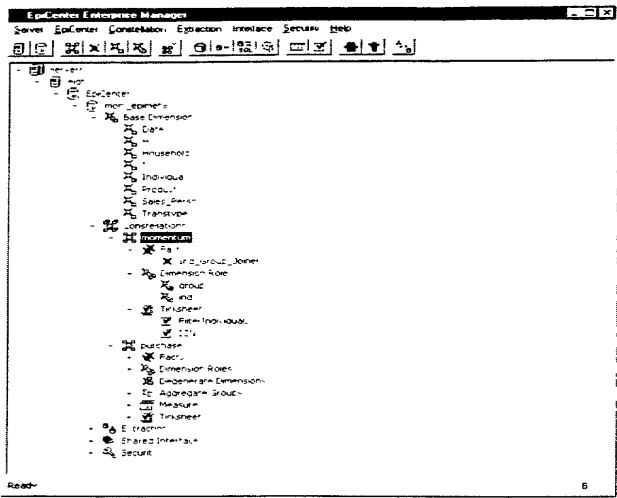


Figure 3: EpiCenter Enterprise Manager Window

Dimension Tables

A *base dimension table* consists of dimension columns that hold the actual attributes extracted from the source system. The Epiphany schema allows dimensions to be shared by all of the constellations within an EpiCenter. This is accomplished through the use of base dimension tables that serve as master tables for the entire EpiCenter. Each base dimension table may be referenced by multiple fact tables, or multiple times by the same fact table. These references are called *dimension roles*.

Dimension Tables

Dimension Roles

A row in a fact table may have several foreign keys that point to the same base dimension table, but the role usage differs. (A *foreign key* is a reference from one table to another table.) For example, a dimension role within an Order constellation and a dimension role within a Customer Support constellation may correspond to the same underlying Master Product List table for data. Within the constellation, these dimension roles may be assigned the role of Order Product List or Customer Support Product List, as appropriate. Multiple roles within a single constellation can refer to the same base dimension.

Dimension roles are defined within a constellation, and any fact tables in that constellation may inherit the dimension role. Dimension roles within constellations point to their associated base dimension table for their data. (A dimension role always has an associated base dimension table.)

Date and Transtype Dimensions

The Date dimension and Transtype (transaction type) dimension are assumed common to all constellations, and therefore are automatically provided as base dimension tables. To ensure consistent treatment of time, Epiphany uses a single date dimension throughout the system.

Transaction type dimensions are necessary because the EpiCenter must be able to distinguish among different rows in a single fact table, such as shipping transactions versus bookings, and bookings versus booked returns.

005270 075548

Attributes

Attributes describe a dimension. Some attributes, such as Customer Name, are free-text descriptions, but most have a discrete set of values. For example, a Territory dimension table may consist of its territory key and text that describes each of the organization's sales regions; such as, Eastern, Western, and Southern. It may also contain attributes for a region's subdivisions, such as City and State. Therefore, a hierarchical relationship exists in the dimension table in which the City attribute rolls up (aggregates) into State, which then rolls up into Region.

Region	State	City
Western	CA	Los Angeles
Western	WA	Seattle

Note: In the EpiMart, facts are quantifiable measurements, usually numbers. Attributes of dimensions are usually character strings.

Fact Tables

Fact tables are physical database tables that contain dimension role foreign keys and fact columns. A fact column is a single column in the fact table whose values are numeric, such as *net_price*. In the EpiCenter, a measure is an arithmetic combination of fact columns.

A row in a fact table represents a relationship among a set of values (each value is a separate field in the table). As shown in Figure 4, a row may exist for a specific customer number, product number, territory number, and date—all foreign keys that point to base dimension tables for their data—and fact columns, such as the number of units and the price per unit. The table may have millions of rows of data.

EPIPHANY SYSTEM OVERVIEW

An Epiphany site's source data may reside in any number of database management source systems, both relational and non-relational. Examples of supported relational database management source systems (RDBMS) and their related software are—

- telemarketing applications (ACT! and Aurum)
- sales force automation (Siebel and Onyx)
- enterprise resource planning (Oracle Financials, PeopleSoft, SAP R/3)
- customer support applications (Clarify and Vantive)

As shown in Figure 5, at the back end of the Epiphany system, data flows from the source databases into the RDBMS server database where the Epiphany database, called the EpiMart, resides. The Epiphany system accesses the source data in a read-only fashion; no changes to an existing source system are necessary. Database updating occurs as part of the database extraction process, which is usually run on a nightly basis.

At the front end, users open a ticksheet in a Web browser on any computer system that supports a JavaScript-enabled browser and select options for the kind of data they want to display. (A ticksheet is a user-interface form in Epiphany that allows end users to construct queries; it is called a *ticksheet* because users select, or tick, items on a page.) The Web browser communicates with the Web server, which runs a Java program that queries the RDBMS server for the relevant information. The report is quickly generated and presented as an HTML document. The Epiphany Application Server is the Windows NT Service that connects with a Web server and delivers Epiphany ticksheets and reports. In the Epiphany system overview, it logically sits between the RDBMS and the Web server.

Epiphany System Overview

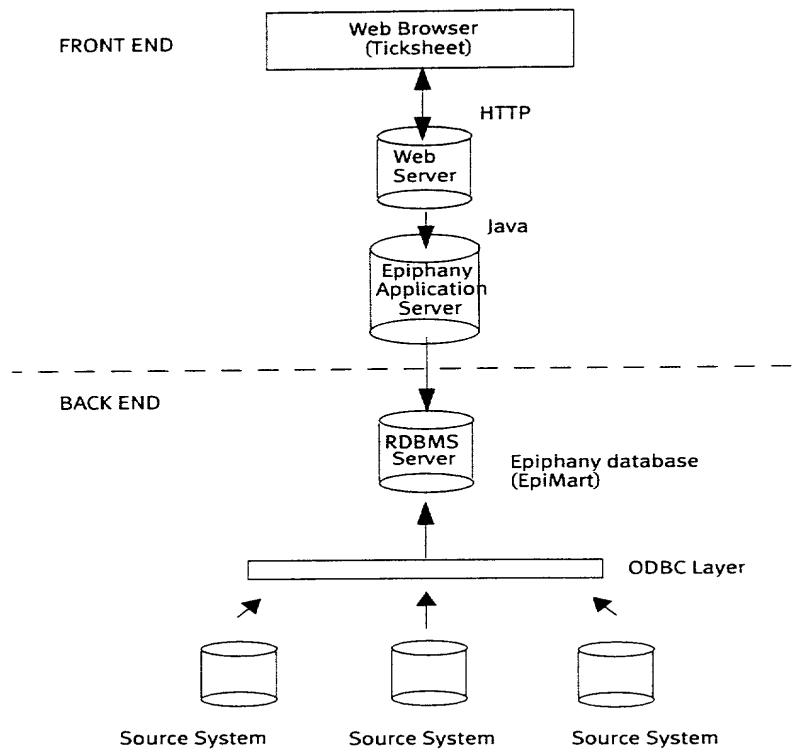


Figure 5: Epiphany System Overview

Although Epiphany's back-end tools can be used to generate and populate an industry-standard star schema, the power of the system derives from its integration with Epiphany's end-user Enterprise Relationship Management (ERM) applications. In order to achieve a robust, consistent application suite, each component interacts with a common metadata repository called EpiMeta.

EpiMeta is built on top of a relational database engine. EpiMeta's tables store all persistent aspects of an Epiphany implementation. When you configure an Epiphany system (using EpiCenter Manager), these are the tables you write to. Each of Epiphany's ERM applications is a consumer of this metadata, which determines the appearance of the applications, as well as other aspects of their behavior.

THE EPIPHANY SYSTEM IS METADATA

All of the control information for an EpiCenter is stored in a single metadata repository called EpiMeta. EpiMeta represents a transactional, fully relational model of over one hundred and fifty tables that have complicated declarative referential integrity constraints. Epiphany provides the EpiCenter Manager tool for configuring this metadata without the need to write to, or even know about, the underlying data structures.

The EpiMeta tables of metadata control all aspects of the system including:

- Star schema structure
- Extraction workflow and semantics
- Aggregation and other performance enhancements
- Business calculations (or measures)
- User interface layout
- Security information
- Saved Report objects

09625518 072500

The Epiphany System Is Metadata

EpiMart contains the customer data at an implementation. The schema of tables in EpiMart is completely determined by the schema metadata in EpiMeta. The contents of EpiMart tables are determined by the instructions contained in the extraction metadata tables (and the relevant contents of the source system). Theoretically, the contents of EpiMeta can be used to reconstruct an equivalent EpiCenter; that is, re-extraction from the source system using the same EpiMeta should result in an identical EpiMart. Because EpiMeta effectively defines an EpiCenter, Epiphany provides an export/import utility to back up metadata, or to transfer subsets of metadata between EpiCenters. (see Appendix G. "Export/Import of Metadata" for more information.)

Each Epiphany tool reads from and/or writes to EpiMeta. The interaction between tools and EpiMeta is described below:

EpiCenter Manager	Writes schema metadata, extraction metadata, security metadata, and aggregate definition metadata. EpiCenter Manager is also used for configuring user-interface metadata, including measures, ticksheets, and dictionary entries. It reads schema metadata and writes measure and ticksheet metadata.
EpiChannel	Reads schema, extraction, semantic metadata. Writes logging data about the extraction.
Aggbuilder	Aggregation, the pre-calculation of selected computations involving facts to speed up the front-end query process, is performed by the Aggbuilder program. Aggbuilder reads schema metadata and aggregate definition metadata, and writes aggregate navigation metadata. <i>Aggregate navigation</i> is the process by which the Epiphany query machinery determines the optimal aggregate table to use to satisfy an application's request for information. (The <i>query machinery</i> is the component of the Epiphany Application Server that communicates

with EpiMart and actually issues SQL statements against the DBMS.)

Epiphany Application Suite

Clarity, Relevance, and Momentum applications read schema, measure, ticksheet, and aggregate navigation metadata. They write saved object metadata.

Each table in EpiMeta defines an integer primary key. The database engine provides the actual unique primary key values for each row in the metadata. All foreign keys between metadata tables are accomplished with these integer columns. The benefit of using non-natural primary keys is that all other columns in the metadata can be changed without affecting relationships in the model. Note that each foreign key uses declarative referential integrity to ensure consistent metadata. Additionally, EpiMeta enforces other declarative integrity constraints based on the correct interpretation of these tables.

THE UNIFORM TREATMENT OF TIME

In many Enterprise Relationship Management (ERM) applications, the time on which a fact occurs is a crucial dimension. It can be complicated, however, to create the semantics of calendars and other date attributes for a new data warehouse. Epiphany provides a time dimension via the special table *Date_0*, along with EpiCenter Manager, for populating the date dimension with the parameters that make sense for each implementation. Each row in this table represents a single day and the attributes associated with that day.

Every fact table in EpiMart automatically contains a foreign key to this special date dimension. Many of the fact semantic templates (the generic programs that perform business transformations on extracted data) depend on the presence of this field. By joining the fact table to the date dimension, Epiphany's ERM applications are able to answer questions of the form:

- Which facts occurred this quarter?
- What is my weekly backlog for the year?
- What products were bought last month?

Uniform Transactional Data

The queries that are constructed for these purposes do not perform date arithmetic, such as:

```
SELECT * FROM MyFact
WHERE date_key > '6/1/1998' and date_key < '7/1/1998'
```

Queries of the above form become unwieldy because of the non-uniformity of the calendar with respect to the number of days in the month, leap years, weekly overlaps with months, and so on. Instead, determining which days belong to which weeks, months, quarters, or years is done once when the date dimension is populated. This table is then fully enumerated with all values of interest for the EpiCenter, allowing queries of the form:

```
SELECT * FROM MyFact, Date_0 WHERE MyFact.date_key =Date_0.date_key
AND cq_and_cy_name = 'Jun 1998'
```

Queries of this form can easily take advantage of RDBMS indexes.

Epiphany allows for specification of Calendar fiscal quarters, as well as 13-week manufacturing calendars (4-4-5 calendars), in which a quarter always starts on the same day of the week. The date dimension can also be configured to specify which weekdays start and end a business week (for example, Sunday through Saturday versus Monday through Sunday).

UNIFORM TRANSACTIONAL DATA

A transaction represents an event in time. All data in an EpiMart fact table is stored in transactional format. For certain business entities, this format makes intuitive sense, such as in the case of an invoice in which a record in EpiMart represents the shipment of a product to a customer at a certain point in time. For the most part, this event cannot be changed; it simply happened and is stored as such in an EpiMart fact table.

Other business entities are not so easily made into transactions. When a customer calls to order a product, he might choose to order 10 units. In the Epiphany system, this fact is entered as a transaction with quantity 10 for Product P1 to Customer C1.

Order Fact

Customer	Product	Date	Quantity
C1	P1	6/1/1998	10

However, if the customer calls back the next day and changes the order to 15 units, then instead of entering a separate transaction of 15, Epiphany's extraction machinery enters the *difference* of 5 as a new transaction on that second day.

Customer	Product	Date	Quantity
C1	P1	6/1/1998	10
C1	P1	6/2/1998	5

Similarly, in the case of Inventory, Epiphany tracks changes in inventory as transactions instead of restating the reported inventory from the outside world. For example, if Customer C1 *reports* inventory for Product P1 as:

Reported Inventory in Source System

Date	Quantity On Hand
6/1/1998	10
6/8/1998	12
6/15/1998	5

09625518.072500

Uniform Transactional Data

In Epiphany's fact table the same information is represented transactionally as:

Inventory Fact

Customer	Product	Date	Change In Inventory
C1	P1	6/1/1998	10
C1	P1	6/8/1998	2
C1	P1	6/15/1998	-7

Of course, an end user who asks for a report of inventory by week wants to see the data reported in Clarity as it is reported by the source system. To accomplish this, use EpiCenter Manager to define a measure that has a backlog type that makes this measure an accumulator over time. This causes the transactions to be reassembled back into the reported format.

Why go to the trouble of disassembling orders and inventories into transactions, only to reassemble the previous format at output time? The reason is that transactions are the most additive way to store data, which means that transactions can be recombined along arbitrary dimensional boundaries.

The way in which Inventory data is reported above can easily answer queries of the form "Inventory by *week*." This data, however, cannot easily be used to answer queries by month or year without some external knowledge of which week is the end of the month. Today's RDBMS engines cannot efficiently handle these types of queries without Epiphany's transactional storage. Note that the inventory for the month of June can be calculated by adding up all the transactions that occur between 6/1 and 6/30, yielding an ending inventory of $10 + 2 - 7 = 5$.

Important: All source system data must be made into transactions. Keep this in mind when you author an Epiphany extractor or choose a semantic type.

ADAPTIVE ARCHITECTURE

In traditional client/server application environments, changes to an underlying table's schema adversely affect programs that operate on that schema. The goal of Epiphany's Adaptive Architecture is to allow on-the-fly changes to the EpiMart schema while preserving the form and proper operation of the entire Epiphany Application Suite.

A schema change in EpiMart can affect these components:

- Extraction statements that populate the staging tables.
- Semantic instance programs that merge staging data with EpiMart data.
- Aggregates defined on these tables.
- Measures that refer to these tables.
- Ticksheets that refer to columns in these tables.
- Security restrictions on columns in these tables.

The use of a single metadata repository (EpiMeta) ensures that all components of the system receive notice of a change simultaneously; for example, Aggbuilder will not try to build aggregates on tables or columns that no longer exist.

During extraction, semantic instances are SQL programs that accomplish business transformations on extracted data. Since these programs must be changed in response to schema changes, these programs are compiled on-the-fly at execution. Thus the program uses the latest schema metadata in its construction, which results in a well-formed set of SQL statements.

EpiCenter's Generate Schema command creates and modifies the EpiMart tables. The first time this operation is executed, all tables are built using CREATE TABLE statements. These tables include base dimension, fact, and external tables. Certain metadata fields are used in the actual construction of these tables.

EpiCenter Manager ensures that these fields follow the proper naming conventions for table and column names. These naming conventions are as follows:

- Base dimensions

Adaptive Architecture

The name of the table is taken from the name of the base dimension. Each dimension column becomes a physical column in the EpiMart table with the same name.

- Fact tables

The name of the table is taken from the name of the fact table. Each metadata fact column becomes a numeric column in the EpiMart table with the same name. Each dimension role and degenerate dimension of the constellation to which the fact table belongs become a foreign key column in the fact table. [A *degenerate dimension* is a single column dimension in a fact table that stores textual information in lieu of using a foreign key to a base dimension table. For more information, see "Degenerate Dimensions" on page 108.]

- External tables

The name of the table is taken from the name of the external table. Each external column becomes a column in the EpiMart table with the same name.

Each time Generate Schema is executed, EpiCenter Manager records the schema of the newly built tables in a set of metadata tables called the Actual tables. When subsequent changes are made to the schema definition, EpiCenter Manager examines these Actual tables to compute a delta operation.

Important: If Generate Schema is not run and the schema definition has been altered, neither Aggbuilder nor the Epiphany Application Server will start. Both will recognize a difference between the metadata schema definition and the contents of the Actual tables.

The following table describes the schema operations.

Table 1: Schema Operations

Operation	Action
Add a new base dimension	The table is created from scratch.
Delete a base dimension table	The table is dropped, and the dimension role columns of any fact tables that point to that base dimension table are deleted.
Rename a base dimension table	The table is renamed.
Add a new dimension column to an existing base dimension	The existing table is altered to include that column. All existing rows take the default value for that column.
Delete a dimension column from a base dimension	The column is removed.
Rename a base dimension column	Treated the same as a deletion of the old column and an addition of a new one.
Add a new fact table	The table is created from scratch.
Delete a fact table	The fact table is dropped.
Rename a fact table	The fact table is renamed.
Add a new fact column	The table is modified to have the new column. All existing rows get the value 0.
Delete a fact column	The column is removed.
Rename a fact column	Treated the same as a deletion of the old column and an addition of a new one.
Add a dimension role	The column is added to all fact tables in that constellation. The value for all existing rows is set to 1, which is a special value that always points to the UNKNOWN row in the base dimension to which that role refers.
Delete a dimension role	The column is removed from all fact tables in that constellation.

Adaptive Architecture

Table 1: Schema Operations

Rename a dimension role	Treated the same as the deletion of the old column and an addition of the new one.
Add a new degenerate dimension	The column is added to all fact tables in that constellation. All existing rows get the special value UNKNOWN.
Delete a degenerate dimension	The column is removed from all fact tables in that constellation.
Rename a degenerate dimension	Treated the same as a deletion of the old column and an addition of the new one.
Add a new external table	The table is created from scratch.
Any change to an external table including changes to column	The table is dropped and recreated.

EPIPHANY DATABASE EXTRACTION

The goal of the extraction process is to extract raw data from data source systems and to place it into EpiMart tables where the data is accessible for query by front-end users. The extraction process, which is usually performed on a nightly basis, involves the use of these components:

- **EpiCenter Manager.** In addition to using EpiCenter Manager to define your site's star schema and constellations, you will use it to define extraction jobs. See "Extraction" on page 177 for instructions.
- **EpiCenters (EpiMeta and EpiMart).** EpiMeta is the metadata database that defines the star schema and the semantics applicable to an organization. (*Semantics* refers to the means by which data is interpreted for an organization in terms of its business processes.) EpiMeta also contains data related to the extraction jobs.

EpiMart consists of fact, dimension, and staging tables. The fact and base dimension tables hold the data of the star schema (actual customer data), and the staging tables help to populate them.

Epiphany Database Extraction

- **Semantics** (types/templates and instances). At a high level, a semantic type reflects the semantics of a company's business processes. During data extraction, semantic instances, which are post-compilation SQL programs, are used to merge data with the proper semantics into data loaded during previous extractions. See "Data Merging" on page 42 and "Semantic Instance Extraction Step" on page 51 for definitions of semantic-related terms.
- **EpiChannel**. This is the Epiphany extraction program that executes jobs and logs job activity. See "Running Jobs: EpiChannel" on page 67 and "EpiChannel Debugging" on page 74 for more information.
- **Database administration tools**. These tools are supplied by the database vendor to administer the physical characteristics of the databases and the interconnections among them. (Please refer to your database vendor's documentation for information about these tools.)

The major topics covered in this chapter are—

- "Extraction Phases: An Overview" on page 41
- "Data Stores" on page 43
- "The Role of Jobs in Extraction" on page 45
- "Extractors" on page 46
- "System Calls" on page 53
- "Aggregate Building" on page 55
- "MomentumBuilder" on page 60
- "MomentumBuilder" on page 60
- "Custom Fact Indexing" on page 62
- "The UNKNOWN Dimension Row" on page 64
- "Running Jobs: EpiChannel" on page 67
- "EpiChannel Debugging" on page 74
- "EpiChannel Output" on page 80
- "Monitoring Jobs" on page 81
- "Mirroring: A and B Tables" on page 83
- "SQL Limits for Facts" on page 84

EXTRACTION PHASES: AN OVERVIEW

There are three main phases of the extraction process—load (pull/push), data merging, and aggregate building. Each phase is related to the metadata that defines the tables and their columns.

Load Phase

During the load, or pull/push extraction phase, data is extracted from a source database or file in raw format and loaded into interim tables (staging tables or external tables). A *staging table* is a table that contains all of the fields required by a single base dimension or fact table. Staging tables hold the data in preparation for the second phase of extraction: data merging.

An *external table* is a table built in EpiMart (and thus internal to the Epiphany system), which usually serves as a temporary table during extraction. The external table, which is similar to a staging table, receives the data directly. External tables serve as intermediary tables when more complicated multi-staged extraction is required.

Staging and external tables have a schema similar to the target EpiMart tables (Epiphany's Adaptive Schema Generator generates both simultaneously). The *Adaptive Schema Generator* is a component of EpiCenter Manager that builds the tables in EpiMart using the schema metadata definitions in EpiMeta.

When you run the EpiChannel program to start the extraction process, it reads the metadata for the job you selected and begins executing extractor steps. An *extractor* logically specifies a series of steps that move data from the input to the output data source. Occasionally, an extractor step is directed to populate external tables rather than staging tables.

Data Merging

Data Merging

The second main phase of extraction is data merging. After the staging tables are fully loaded, semantic instances merge the new data in the staging tables into the EpiMart database tables. A *semantic instance* is the usage of a generic semantic type on one fact or dimension table during part of an extraction job. As part of defining tables, you assign each an Epiphany-defined semantic type. A *semantic type* refers to the logical business process for which a generic SQL program called a *semantic template* will be applied.

The Epiphany extraction program, EpiChannel, executes the semantic instance at the appropriate time during an extraction job.

Aggregate Building

The third and final extraction phase is aggregate building. Aggregates are pre-calculated and pre-stored summaries of data that significantly accelerate the front-end *query machinery*. The query machinery is the component of the Epiphany Application Server that communicates with EpiMart and issues SQL statements against the RDBMS.

You will use EpiCenter Manager to define which facts are aggregated for groups of dimensions. The aggregate builder program (**agg.exe**), which typically runs immediately following the data extraction, performs roll-ups, or aggregations, on the groups and includes them as aggregates in the EpiMart tables.

At the conclusion of the extraction phase, the extracted data resides in the proper tables and columns in the EpiMart. The data has been extracted from the source system or file data source (called a data store) and placed in the data store on the RDBMS server.

DATA STORES

A *data store* is a logical location of data that functions either as a source or sink within an EpiCenter. Data stores include relational database connections, as well as files and directories. The concept of a data store is integral to the Epiphany extraction process. A data store represents the physical location of a source of data that the Epiphany system can read from and/or write to. Typically, the input data store is a source system database or file, and the output data store is EpiMart, which holds customer data, on an RDBMS server. As part of the installation procedure, you will create a new, empty database for EpiMart (and EpiMeta).

As shown in Figure 6, the Epiphany system extracts the raw data from a site's source system—for example, a RDBMS server, a generic ODBC source system, or a flat file—via Open Database Connectivity (ODBC). ODBC is an abstraction layer that provides a common interface to many databases. This data is read by various ODBC drivers and written using the target database system's API. The Epiphany system can access data if an ODBC driver exists.

EpiChannel, the Epiphany extraction program that executes jobs and logs job activity, opens an ODBC connection to the source system and a database library connection to the target server and initializes tables.

You will use EpiCenter Manager to configure the data stores as part of setting up your site's extraction process. When an EpiCenter is created, it has three default data stores:

- *EpiMart*
A special data store for the actual database on which Epiphany's front-end applications will run.
- *JobFileLog*
Specifies the data store for EpiChannel job logs.

Data Stores

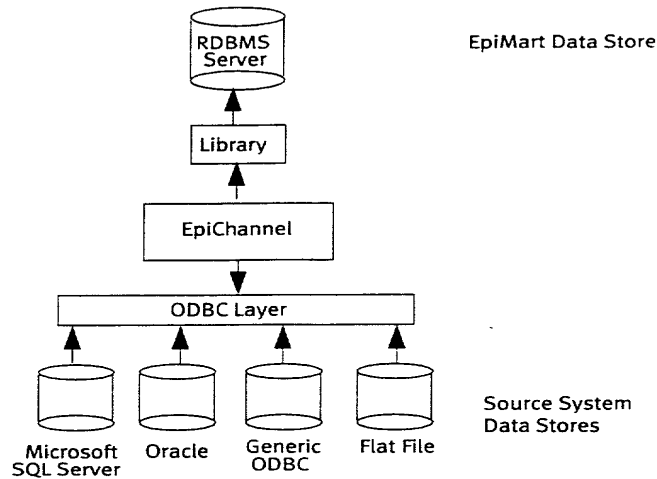


Figure 6: The Back End of the Epiphany System

- **LoggingDB**

Specifies the data store for the EpiChannel logs.

You may, however, set up as many data stores in EpiCenter Manager as necessary. For example, operational systems for different divisions within an EpiCenter may have the same fact and dimension tables (EpiMart), but draw data from different source systems. In this case, you would create a data store for each input source system. For more information, see “Data Stores” on page 177.

Note: The EpiMart is the data store, or database, that you set up for your data warehouse. The EpiMeta holds the description of your data (the extraction jobs, the ticksheets, and the star schema definitions). The EpiCenter is a combination of EpiMart and EpiMeta.

THE ROLE OF JOBS IN EXTRACTION

The Epiphany extraction process works through the running of jobs. A site defines the jobs needed to extract its data and runs these jobs on a regular basis to update the EpiMart. A *job* is basically a batch of work to be performed as a unit. The work consists of an ordered list of job steps, each of which is either an extractor or a system call. System call job steps may be interspersed with extractor job steps in any order. (See Figure 7, on page 45 and Figure 8, on page 47.)

A simple extraction may consist of one job, which has multiple steps. Each step utilizes the data produced by the previous step. A multi-stage extraction, however, may consist of multiple jobs. The EpiCenter Manager provides a graphical user interface (GUI) for defining job steps and the job input and output data stores. The actual job steps (extractors and system calls) need to be customized for each site.

After the jobs have been defined and the EpiCenter schema generated, the database administrator can invoke the EpiChannel (**extract.exe**, or **extract**) command to begin the extraction process that will populate a new EpiMart, or update an existing one.

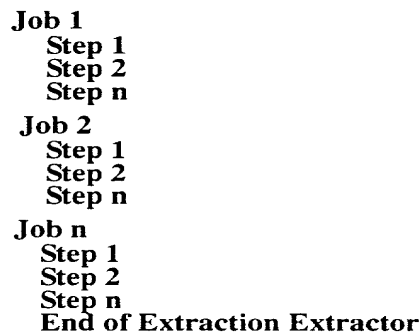


Figure 7: Job Work Flow

EXTRACTORS

An extractor is one or more sets of SQL operations that will be executed against a particular source and destination data store. These same sets or groups of SQL operations may be shared by another extractor that applies them to different data stores. The SQL operations may be either SQL statements or semantic instances. In general, SQL statements are used to extract data, and semantic instances are used to merge data with the proper semantics into data that has been loaded during previous extractions.

Semantic instances are designed to work on an idealized star schema. Macros in these semantic instances are translated at extraction time to the tables and columns in the target star schema. (The fact or dimension table listed in the Semantic Instance dialog box (Figure 53, on page 188) in EpiCenter Manager determines which column names are actually used in the final SQL.) This enables the complex transformation logic in a semantic instance to be applied to star schemas other than the one for which the semantic instance was first used.

As shown in the job work flow (see Figure 8), a job step may be either a system call or an extractor. An *extractor* is a list of extractor steps (and input and output data stores) that move data from the input to the output data store. An *extractor step* is a single step of an extractor, which always points to an extraction group. An *extraction group* is a set of extraction steps. An *extraction step* is a single atomic extraction operation that can be either a SQL statement or a semantic instance.

Extraction steps and groups are both modular. After you define them, you may reuse them in other jobs. (The Extractor Steps dialog box (Figure 51, on page 183) in EpiCenter Manager is where you define a new, or select an existing extractor for a job.)

Note: An extractor has associated input and output data stores and exists independently of any extraction step.

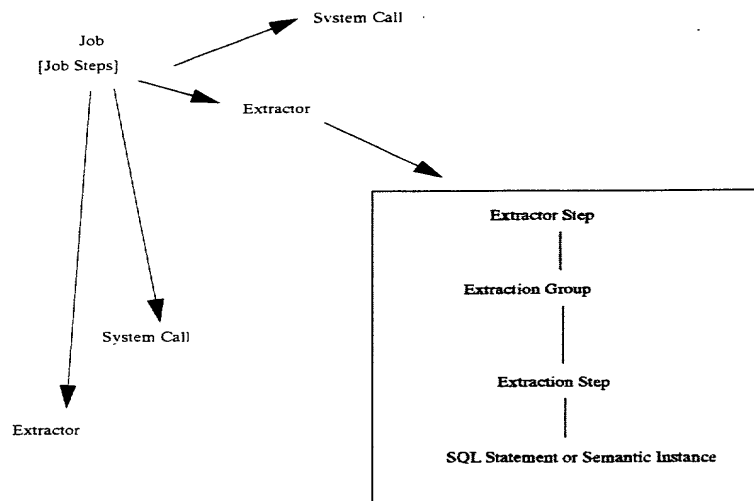


Figure 8: Job Structure

An extractor can move data from a single source data store to a single destination data store. A job may consist of multiple extractors, however, each with its own independent input and output data stores.

Jobs may also share extractors, applying the same logic in different orders—if the jobs' data stores have common structures. For example, a job that runs Monday through Friday could share the same extractor with a Weekend version of the same job in which the only difference is that the Weekend job has an extra step, an extractor that updates rarely changing tables.

The two kinds of extraction steps that may comprise an extractor step are discussed below: SQL statement and semantic instance.

SQL Statement Extraction Step

An SQL statement extraction step may be either a stand-alone SQL statement or a pull/push statement.

A stand-alone SQL statement is executed against either the source or destination data store for that extractor and is usually specific to the data store's type. These SQL statements are usually issued to achieve a side effect, such as dumping the transaction log or setting up environment variables. Any returned results are discarded.

A pull/push SQL statement is a type of extraction step that issues SQL against an input data store (and is interpreted according to the rules of that database's engine). The returned results are inserted (pushed) into the destination table for that step, either a staging table or an external table (the output data store of the extractor).

The result set columns are mapped either to columns in the destination tables or to other functions. The mapping is a case-insensitive exact match by name of the available columns, or function results, to the destination table columns.²

At this stage of the extraction, each destination table column must have a matched value. If not, the mapping fails, the SQL statement results are discarded, and the statement fails. Extra columns that are available from the SQL statement or functions, but which are not used in the destination system, are not considered to be errors. (SQL statements with extra columns will have their mapping displayed in the EpiChannel log if you set the verbosity level for EpiChannel high enough; see "EpiChannel Debugging Levels" on page 78.)

Once mapped, the rows from extraction SQL statements are fetched into EpiChannel memory where they are verified for field width and forwarded to any function that consumes the field. The mapped results are forwarded to the destination database.

² Special characters such as the at symbol (@) are removed when this match is performed.

SQL Statement Extraction Step

External Tables

Extraction statements are sometimes directed to load external tables rather than staging tables. External tables serve as intermediary tables when more complicated multi-staged extraction is required, such as for the grouping of data, dividing values in a field into bins, or joining with data from another source system. They may also serve as holding fields used by non-Epiphany data transformation tools, such as third-party name and address cleansing tools. In the case of cleansing tools, extractors merge the cleansed names and addresses with other tables extracted from the source databases into the EpiMart tables.

EpiCenter Manager provides the extractor named *End of Extraction* for each EpiCenter by default. This extractor consists of two steps:

Step 1: Issue SQL against the EpiMart to determine the date of the last extraction to be displayed to users.

Step 2: Toggle the current EpiMart from A to B, or vice versa.

Step 1 usually consists of SQL statements that load the external table named *last_extract_date*. The *last_extract_date* table is provided by Epiphany, and is recommended for use at your site.

Step 2 updates the *config_master* value *last_extract_date* with the latest value received from Step 1. It then updates the *config_master* value called *current_datamart*, switching it between A and B. (*config_master* is a metadata table that contains various system parameters in name/value format; these are modified via the Configuration dialog box in EpiCenter Manager.)

Note: The last extraction date setting is the date that appears as the *Data is valid as of...* in Clarity reports. It is determined by an option in the EpiCenter Manager's Configuration dialog box (described in Appendix B, "EpiCenter Configuration").

Switching the focus between the A and B tables involves an update to a single row in the database (no tables or databases are actually moved). See "Mirroring: A and B Tables" on page 83 for a discussion of A and B tables.

See "Using External Tables as Inputs to Staging Queries" on page 310 for more information about external tables.

All operations on the source data store or the EpiMeta data store occur through the appropriate native library (API) for the EpiCenter. The inserts are batched so that the extraction statements maintain a count of rows fetched, rows forwarded, and rows committed. The batch is fully sent and committed before the extraction SQL statement ends and the next job step may begin.

Any error, as determined by the database engine evaluating the SQL, results in the SQL statement being considered in error. If the action plan for the SQL statement (as selected in the General tab of the SQL statement dialog box) is to abort, then the job halts without executing any other SQL operations or job steps.

SQL Macros

A SQL statement may contain extraction macros that are expanded before the SQL is executed. Some of these macros provide database-independent functionality by expanding in different ways for different databases, and some use the metadata to cause the SQL to reflect the edits made in EpiCenter Manager. These macros are described in Appendix A, "Epiphany Macros."

Staging Tables

The most common destination for an extraction statement is a staging table. As mentioned, a staging table contains all of the fields required by a single base dimension or fact table. Unlike the real fact and base dimension tables in the EpiMart, the staging tables have dimension columns that contain key fields from the customer's system, rather than Epiphany-generated foreign keys. Staging tables typically contain data extracted by only one job, rather than accumulating data over time.

Once a staging table is populated, semantic instances are used to merge the new data in the staging table into the existing data. A semantic instance is a series of statements, similar to a program. These statements maintain the integrity and additive nature of the columns in the facts tables and maintain correct references to dimensions even though the dimension may change from one reporting of a fact to another.

See Appendix E, "Writing Staging SQL Statements" for more information.

Semantic Instance Extraction Step

As mentioned, the Epiphany extraction process (exclusive of aggregation) is a two-phase operation:

- load (pull/push) phase
The load phase refers to the loading of staging tables, which are the first entry points of data into EpiMart.
- data merging phase
The data merging phase refers to the loading of staging tables into the EpiMart's permanent base tables.

This process of moving data from staging tables into the EpiMart's base tables is called *semantic transformation*. Proper configuration of the data merging phase of extraction requires an understanding of the underlying meaning, or semantics, of the data.

Epiphany provides a set of semantic types/semantic templates that you may use without having to be concerned with their internal code. A *semantic template* is a generic SQL program intended to accomplish specific business rules during extraction (these rules are referred to as *semantic types*). A semantic template program contains SQL with generic table names; it does not refer to actual column or table names. Only when the semantic template is applied to a base dimension table or fact table does the SQL contained within it refer to actual column and table names.

When a semantic template is combined with a base dimension table or a fact table, the result is a semantic instance. A *semantic instance* is a valid SQL program that can be run against EpiMart.

Applying a Semantic Type

While SQL Statement extraction steps are used to move data from the source systems (non-Epiphany data stores) into temporary staging tables in the Epiphany datamart, semantic instance extraction steps move data from staging tables into fact tables and base dimension tables, where the data can be queried by the end user.

Applying a Semantic Type

To define a semantic instance extraction step, you will use the Semantic Instance dialog box in EpiCenter Manager to assign a semantic type to either a base dimension table or a fact table. Semantic types are rules provided by Epiphany that use business logic to regulate how new data from a staging table is merged with existing data in the base dimension table or the fact table.

The simplest semantic types (Initial Load Dimension and Initial Load Fact) simply overwrite the base dimension table or the fact table with the contents of the staging table; other semantic types deal intelligently with backlogs, changes in dimensions over time, and so forth when combining new and existing data. The different semantic types are described in Appendix F, "Semantic Types."

You must use at least one semantic type for each base dimension table and fact table in your EpiCenter during extraction. Otherwise, new values for those base dimensions or facts that lack semantic types will not make their way into the EpiMart. You do not usually want to use more than one semantic type for a single base dimension or fact table since the semantic types generally cancel one another out. The exception is that certain semantic types for facts (Count Unjoined and Custom Fact Index) do not actually merge data from the staging table into the EpiMart. These semantic types have no effect on the base dimension table or fact tables and are instead useful for their side effects. These special semantic types should be used in conjunction with other semantic types that perform the actual merging of data.

Dimension semantic types have two purposes: creating base dimension tables and creating dimension mapping tables. Dimension mapping tables are used to convert dimension source system identifier keys (*sskey*) in fact tables to Epiphany dimension keys. Dimension semantic types are also responsible for proper indexing of these two tables.

Fact semantic types are normally used to populate and Index new fact base tables. See Appendix F, "Semantic Types" for a description of source system keys.

SYSTEM CALLS

While SQL-based transformations may be sufficient to extract from simple databases, sites with more complex databases may require multi-stages and additional commands, such as lookup tables, aggregation splits, gathering data into ranges (binning), and duplicate detection. For this purpose, you may use system calls, which are executed during a job as if invoked from the console's DOS command line.

In a multi-stage extraction, files might need to be moved, decompressed, de-encrypted, or purged. Although these steps could be placed before or after extraction statements, they may need to be placed in-between SQL-based steps. EpiChannel coordinates the execution of system calls between extractors.

A complex job might require the following steps:

- Step 1:** The invocation of a system call to uncompress a file.
- Step 2:** The invocation of an extractor to populate external tables.
- Step 3:** A system call to invoke third-party software to cleanse the names in the external tables.
- Step 4:** Extractors to merge the cleansed names (along with data extracted from other tables in the original database and other databases) into the EpiMart.

Many system calls need to reference the location of databases or files. In the above examples, Steps 1, 2, and 3 need to share the file and database names for the data they pass to each other. Although these names could be hard-coded, this is not the best approach; for example:

- The system call could break if the database login information changed, and the system call text was not modified accordingly.
- The system call cannot be easily reused.
- Some file names need to be adjusted from run to run so that useful intermediate data is not overwritten.

System Calls

Job data store roles provide an alternative to hard coding database and file information in system calls. Roles are referenced by macros in systems calls, and these macros look to the job to see what data stores are associated with the roles. The information from the matching data store is replaced by the macro. For example, the system call

```
mkdir $$DIRNAME[Working Dir]/aggbuilder_temp
```

would have the `$$DIRNAME` macro expanded to the MS-DOS file path that EpiChannel is using as a *working directory*. The **mkdir** command then creates a subdirectory in this location. Because EpiChannel generates a unique directory name for itself for each run, the above command would create a corresponding unique subdirectory for use by the Aggregate builder program (Aggbuilder), and this directory would be moved/purged along with EpiChannel logs.

The system call is executed from the working directory in the same way as if you opened a *command.com* window and changed directories to that location. You should either use an absolute path to specify the command, or you should make sure the value of the PATH environment variable includes the directory that contains the command executable.

`$$EPIBIN` expands to the path of the **extract.exe** directory, so if you enter:

```
"$EPIBIN/mycommand.exe" arg1 arg2 ...
```

and place **mycommand.exe** in the same **win32** directory as **extract.exe**, then it will work. If you do anything else, you are subject to the rules of DOS.

Note: Use quotes as shown for "`$EPIBIN/mycommand.exe`" `arg1 arg2 ...` because DOS interprets the file name line, which is

C:\Program Files\Epiphany, as two *command.com* arguments unless the file name line is placed within quotes.

See Appendix A, "Epiphany Macros" for more information about Epiphany macros.

Note: All system calls are expected to have an exit code of zero, but you can use the Add Job Step dialog box (see Figure 56, on page 193) in EpiCenter Manager to set the *On Error* type to Abort or Ignore, as appropriate.

AGGREGATE BUILDING

Creating aggregates speeds up system performance at the front-end of the Epiphany system. This is because some queries simply require summary data, which can be pre-computed by the system rather than being recomputed for each query. For example, assume that a site has 5,000 products categorized by ten product types. Clarity users may request reports that show data aggregated by product type. If there is a grouping by product type, the Epiphany aggregation program creates a table with ten rows (one for each product type) and pre-calculates “rolled-up” (combined) fact aggregates based on this smaller dimension table. By using significantly smaller fact aggregates, the system can generate reports considerably faster.

When EpiMart is successfully populated, the rows in the fact tables contain foreign keys to dimension table rows. In Epiphany, all dimension primary keys are stored as integers. For example, fact rows may resemble those in fact table Order_0, whereas dimension rows may resemble those shown in Customer_0, Product_0, and Salesperson_0.

Order_0 Fact Table

Customer_key	Product_key	Salesperson_key	Total Sale
6	8	5	\$20
5	6	6	\$30

Customer_0 Dimension Table

Customer_key	Customer Name	Region
5	ABC Company	West
6	Bill's Lighting	West
7	Fred's General Store	East

09625518.072500

Aggregate Building

Product_0 Dimension Table

Product_key	Product Name	Platform
6	Product A1	Windows
7	SuperX	Mac
8	Smash-Em	Windows

Salesperson_0 Dimension Table

Salesperson_key	Salesperson Name
5	Gene
6	Sue

All fact and dimension tables that end with **_0** (underscore zero) are called base tables in EpiMart. They contain raw data at the level that is extracted from the source system.

A base dimension is a physical dimension table in EpiMart, such as Customer or Product. Base dimension tables contain the actual dimensional values that are available for reporting or filtering. Attributes, such as Customer Region or Product Platform, can be rolled up. For example, end users often want to create reports of total sales by region. Suppose that Order_0 has hundreds of thousands of records, and Customer_0 has many thousands of records, and that there are only three different regions in the Region field. A report of sales by region against Order_0 and Customer_0 will need to aggregate over thousands of records, causing the report to return answers very slowly. However, a pre-built aggregate with three (rolled-up) rows that already has the total sales by regions could answer the same question very quickly.

After configuring an EpiCenter, the following aggregates might be built:

Order_17

Customer_key	Total Sale
1	\$50

Customer_1

Customer_key	Region
1	West
2	East

Note the following:

- The *Customer_1* table contains only the rolled-up field(s). The number of rows has been reduced so that each row contains a distinct Region. The table name is the same as the base table, with a different number at the end.
- The *Order_17* table contains the same total sales dollar amount, but the number of rows has been reduced (in a larger example, this reduction could be extremely significant). The name of the table is the same as the base table, with a different number at the end.
- The foreign key *Customer_key* has the same name as in the base tables above, but the numbers differ. It is important to recognize that the *Customer_key* field in *Order_17* can be joined only with the *Customer_1* table, since *Order_17* is at the granularity of Regions, not Customers.

See “Defining Dimension Aggregates” on page 102 for a description of how tables such as *Customer_1* are built using EpiCenter Manager.

The Aggbuilder Program

The Aggbuilder Program

Note: Aggregate building is performed by Aggbuilder (**agg.exe**), a program similar to EpiChannel. (EpiChannel is the program used to populate your initial EpiMart and to update it on a regular basis; see “Running Jobs: EpiChannel” on page 67.) Aggbuilder uses the same command line, EpiMeta database, and job definition table as EpiChannel. Aggbuilder, however, completely ignores the job steps (that is, system calls and extractors). It uses the job table only for an indication of the *log database* and *working directory*. The structure of Aggbuilder log files is similar to that of EpiChannel except that the *log* directory name is *agg_timestamp* instead of *chn_timestamp*.

The easiest way to invoke aggregate building is via a system call that uses some of the system-call macros to isolate details. To invoke Aggbuilder, add a system call similar to this after the extraction and semantic instances steps, but before any use of the extractor called *End of Extraction*:

```
$$AGG $$EXC_ARGS -j agg_job_name
```

where **agg_job_name** is the name of the job with the proper logging information.

To log to the same places as the current job, use a system call such as:

```
$$AGG $$EXC_ARGS -j $$JOBNAME
```

The Aggregate Building Process

When the Aggbuilder program is executed during an extraction job, the following actions are taken:

Note: All actions occur in the set of mirrored tables (A or B) that is *not* currently active. See “Mirroring: A and B Tables” on page 83 for more information.

- Step 1:** Dimension aggregates are built. Each column set for a base dimension will become a dimension aggregate as long as that column set is included in an aggregate group via EpiCenter Manager’s Aggregate Group dialog box (see “Aggregation and Aggregate Grouping” on page 115). Indexes are built for these tables as appropriate.
- Step 2:** For each fact table, all enabled aggregate groups to which that fact belongs are examined. Aggbuilder determines all possible combinations to be built based on the dimension role definitions and creates unique table names. Fact aggregates are physically built and indexed.
- Step 3:** Aggbuilder records what it has built in a set of read-only metadata tables in EpiMeta. You may use EpiCenter Manager to browse these lists. The Epiphany Application Server uses these same lists at runtime to decide which aggregates to use to satisfy an end-user query.
- Step 4:** All activity is logged to Epiphany’s logging tables.

MomentumBuilder

MOMENTUMBUILDER

MomentumBuilder is the executable that creates special Momentum tables, such as mini-dimension tables and clusters, which are described in the discussion on Momentum in Chapter 3. The order of the extraction for a Momentum EpiCenter is as follows:

Step 1: SQL/Semantics

Step 2: Aggbuilder

Step 3: MomentumBuilder

Step 4: End of Extraction

In a Momentum EpiCenter, **EpiMgr** creates a default MomentumBuilder job step that runs the MomentumBuilder program (**MomentumBuilder.exe**) with the appropriate arguments. Similar to Aggbuilder's **agg.exe**, **MomentumBuilder.exe** is usually run as a system call from an EpiCenter Manager job. You may also invoke it from the command line (for example, when you run MomentumBuilder as the sole extraction process).

The **MomentumBuilder.exe** command-line arguments are as follows:

```
-h
-?
-INSTANCEDIR
-EPIBINDIR
-S server_name
-D db_name
-U username
-P password
-v verbosity level
```


A given Epiphany query uses only a *single* fact table (either base or aggregate). Typically, as a user drills down into a result set, the physical table that services each query moves from a high-level aggregate (with few rows) to a larger aggregate (possibly using an index on the aggregate). For the user's response time to be acceptable in each of these cases, you must properly configure both aggregates (as described in the previous section) and custom indexes.

All aggregates are indexed in the same manner as the base table (insofar as the aggregate has the same dimensionality as the index). For example, if the base table has an index on *customer_key*, *product_key*, and *territory_key*, then an aggregate that contains only the Customer and Territory dimensions will have an index on *customer_key* and *territory_key*.

Note: If two different indexes on the base table result in the same index on the aggregate, then the aggregate index is built only once.

By default, each fact table has one index built on its dimensional keys during extraction (on Microsoft SQL Server this index is also the CLUSTERED index). For all fact tables, the leading term of the index is *date_key* because a wide class of queries filters on the time dimension. For example, if a system contains data for multiple years and a user asks for a single calendar month, the database almost certainly will use an index when it services this query (whether via the base table or an aggregate). All other dimension roles in the fact table are also included in this index. The order in which they appear is determined by their order in the constellation within EpiCenter Manager's directory tree. To re-arrange this order, right-click a dimension role folder, and select Up or Down from the pop-up menu.

A custom index is the metadata definition of indexes to build on fact tables in EpiMart. Normally, each fact table is given a single index. You can use EpiCenter Manager (the Custom Index tab in the Fact Table dialog box) to:

- configure all other indexes on the fact table.
- assign each index a logical name, although the name is not used when building the physical index.
- choose the dimension roles that make up the index, and their order.

Note: You will use the Custom Fact Index semantic type to actually build the indexes. (See Appendix F, "Semantic Types.")

The UNKNOWN Dimension Row

It often makes sense to build several different custom indexes, each with a different dimension role as the leading term. If, for instance, end users often filter on attributes of the Customer and Product dimensions, then you could build a custom index on the combination (*customer_key*, *product_key*) and another on *product_key* by itself. There is no need for the second index to also include *customer_key* because the first index could easily service joint customer/product queries.

THE UNKNOWN DIMENSION ROW

When a fact staging table is loaded during an extraction, its dimension source system key columns (columns whose names end with *sskey*) normally refer to rows in the base dimension staging tables. (The column name being referred to in the base dimension staging table also ends with *sskey*.) These source system keys are the actual values used to form the relationships on the source system between the fact and dimension source tables.

Some source systems do not enforce referential integrity, however, which may result in source system fact tables that have “dangling references,” or references to non-existent dimension values. Another possible source of a dangling reference is an optional or null foreign key in the source system. In EpiMart, all fact dimension keys must point to valid base dimension entries; and there should be no dangling references.

To prevent dangling references, each base dimension in EpiMart is populated automatically with a single special row called the UNKNOWN row. This row is always available for mapping fact dimension keys that would otherwise not point to a valid base dimension row.

where:

- h** and **-?** Display on-line help.
- INSTANCEDIR** The path to the root directory of the Epiphany instance where MomentumBuilder will run.
- EPIBINDIR** The path to the **win32** directory in the instance (this is the directory provided by the EPIBIN macro).

Either INSTANCEDIR or EPIBINDIR must be specified so that **MomentumBuilder.exe** can locate the **classes** directory in which the MomentumBuilder Java file has been placed by the installer.
- S** Database server name.
- D** Database name.
- U** Database user name.
- P** (Optional) Database password. If omitted, uses a null password.
- v** (Optional) Verbosity level.
- NOMIRROR** (Optional) Runs MomentumBuilder on the currently active datamart.

The default MomentumBuilder job created by the EpiCenter initialization SQL runs **MomentumBuilder.exe** using the EPIBINDIR argument, which is available under Remote Administration installs where there is no *instance_name*.

Verifying MomentumBuilder Extraction

Alternatively, on the command line you may specify **-I** followed by an instance name. In this case, **MomentumBuilder.exe** retrieves all necessary directory locations and database login information from that instance's key in the Registry. This approach is not used by default because it will fail in Remote Administration installs in which the Instances Registry key is not present.

Verifying MomentumBuilder Extraction

To determine if MomentumBuilder extracted the appropriate number of individual and groups, you need to figure out how many rows there should be in the Momentum tables. The tables *all_groups_* and *all_individuals_* are derived from *ind_group_joiner*.

- *all_groups_* contains the real keys that correspond to all of the distinct *group_key* values in *ind_group_joiner*.
- *all_individuals_* contains the real keys that correspond to all of the distinct *indiv_key* values in *ind_group_joiner*.

The size of the *all_individuals_* table should equal the number of rows in *ind_group_joiner* where the *indiv_key* is not equal to 1.

Please refer back to this section after you have read the Momentum-related material in Chapter 3.

CUSTOM FACT INDEXING

EpiCenter supports two methods that dramatically improve query performance: indexing and aggregation. Each accelerates different classes of queries.

In general, the presence of aggregate tables accelerates queries that answer high-level business questions, such as Sales by Business Unit and Fiscal Year. Deep drill-down queries, or highly filtered queries that ask for a small subset of data, are typically enhanced by an index. A fact base table is usually the largest table in EpiMart. Even tables with millions of rows can be accessed quickly via an index when only a small number of data pages is needed to service a query.

Referring to the UNKNOWN Row during Extraction

When writing fact staging SQL statements, you can refer explicitly to this special row by populating the *sskey* dimension role column with the special string UNKNOWN. For instance, on a Microsoft SQL Server source system, you might use the ISNULL() function as follows to translate all null values to UNKNOWN:

```
SELECT
    ...
    ISNULL(cust_code, 'UNKNOWN') customer_sskey
    ...
FROM
    ...
```

Referring to the UNKNOWN Row during Extraction

For a base dimension such as Customer that has dimension columns *customer_name* and *region*, the UNKNOWN row has actual values similar to any other row. By default, each dimension column assumes the literal value UNKNOWN for each of its textual attributes. However, you may use EpiCenter Manager to override this default value so that end-user queries return a different value. To do so, enter a literal string (without quotes) in the Dimension Column dialog box (accessible from the General tab of the Base Dimension dialog box shown in Figure 16, on page 100). Your string will be used instead of UNKNOWN.

The CountUnjoined Semantic Type

While extracting a fact staging table, one often does not know whether the values that appear in the dimension foreign key columns actually refer to dimension rows. While the technique described above for converting null values to the UNKNOWN string works for missing dimension foreign key values, it will not work when a dangling value appears. If no action is taken, then these fact rows will be lost during extraction since the first step of most fact semantic types involves an inner join between the fact staging table and the dimension mapping tables (see Appendix F, “Semantic Types”).

09625518.072500

Normal Extraction Order

The CountUnjoined fact semantic type is specifically geared towards converting these dangling fact references to the special UNKNOWN value. Scheduling this semantic type (after the fact staging table has been loaded) during an extraction will cause a scan of the fact staging table with joins to each of the dimension mapping tables. All *sskey*'s in the fact staging table that do not map to extracted dimension rows will be converted to UNKNOWN.

Note: Execute the CountUnjoined fact semantic type before executing any other semantic types on this fact staging table.

In general, it makes sense to execute semantic types for base dimensions before any of the semantic types for facts (including CountUnjoined) because the base dimension semantic types produce the mapping tables needed by fact semantic types (see "Normal Extraction Order" below).

Normal Extraction Order

Epiphany's recommended sequence of events during an extraction is as follows:

- Step 1:** Load fact staging tables.
- Step 2:** Load base dimension staging tables.
- Step 3:** Execute base dimension semantic Instances.
- Step 4:** Execute fact semantic instances.

The reason for loading the fact staging tables before base dimension tables is subtle. If Epiphany is extracting from a live source system, then new dimension and fact rows could be created while the extraction occurs. If dimensions were extracted first, then when facts are extracted, a new dimension row (such as a new Customer) might have been created, but was missed. If a fact that refers to that new Customer is then extracted, the fact row shows up in the Epiphany system with an UNKNOWN Customer value (because that Customer row was missed by the extraction). By extracting the facts first, the system might not receive the fact row in this extraction, but will retrieve it in the next extraction.

RUNNING JOBS: EPICHANNEL

EpiChannel is the Epiphany extraction program (**extract**) that you will run to populate your initial EpiMart and to update it on a regular basis.

EpiChannel first opens an ODBC connection in the source system and then opens a database library connection to the target system and initializes tables. It establishes start and stop date time boundaries and replaces certain special strings. After replacing these strings, EpiChannel executes a user-defined sequence of SQL statements or system calls.

The SQL is pre-processed to match warehouse and database vendor syntax. System calls are pre-processed to supply directory locations and database login information.

Before attempting the first job step, EpiChannel verifies the availability of databases and tables (if these options are selected in EpiCenter Manager's Job dialog box) and expands most macros. If these fail, the job does not run.

The job will halt when it encounters the first error in any step, unless that step has an error code set to print or ignore the error instead of to abort it. Error codes are set through EpiCenter Manager. You will use the SQL Statement dialog box (Figure 52, on page 185) for extractors, and the Job dialog box (Figure 54, on page 190) for system calls.

Whenever a job succeeds or fails, e-mail notification is automatically sent to the database administrator (or any designated list of e-mail recipients). See "Configuring E-Mail" on page 196 for more information.

The EpiChannel Command Line

This section describes the command-line syntax you will use to invoke EpiChannel at the console's DOS prompt. These command-line parameters indicate the database with the job information (that you configured using EpiCenter Manager) and the parameters related to interactive debugging. Command-line parameters may also specify the initial debugging level, the maximum selection limit, or that the jobs are to be trial runs.

The EpiChannel Command Line

To invoke EpiChannel, enter the **extract** command on the console's DOS command line followed by the parameters described below.

Note: EpiChannel inherits the case sensitivity of the database engine.

```
extract      -h  
              -?  
              -v integer_verbosity_level  
              -I instance_name  
              -S server_name  
              -D db_name  
              -B db_vendor_name  
              -U username  
              -P password  
              -t trial run  
              # row_count  
              -j job_name
```

where:

- h and -?** Display detailed on-line help.
- v *integer_verbosity_level*** Runs the EpiChannel job in verbose mode, which displays the SQL on the screen. See "EpiChannel Debugging Levels" on page 78 for more information.
- I *instance_name*** Specifies which Registry instance key to use. This parameter is mutually exclusive with many of the other parameters because it causes them to be read from the Registry.
- S *server_name*** Specifies the name of the server where the EpiMart resides.
- D *db_name*** Specifies the name of the EpiMeta database.

- B db_vendor name** Specifies the name of the EpiMart database vendor.
- u username** Specifies the username for the EpiMeta database.
- P password** Specifies the password for the user of the EpiMeta database.
- t** For testing purposes, this is the trial-run option that simulates execution of the SQL on the source and destination databases.
- # row_count** Specifies the maximum number of rows to fetch on any SQL statement.
- j job_name** Specifies the name of the Job as defined in the EpiMeta.

EpiChannel Registry Keys

The EpiChannel program uses the Registry keys shown in EpiChannel Registry Keys, which are located in the following Registry path on the local machine: **Software\Epiphanys\Instances\instance_name**.

In the table below *Default Instance* is the string value for Default in the Instances Registry key under the Epiphany Registry key.

Table 2: EpiChannel Registry Keys

Key	Value
1	Default Instance
2	[Default Instance]/DBVendor
3	[Default Instance]/Database
5	[Default Instance]/Server
6	[Default Instance]/Username
7	[Default Instance]/Password

0925518.072500

09/25/18 07:25:00

Output Files

By default, EpiChannel reads the Default Instance key and uses its value to open the Registry tree that holds all initialization parameters. You can override this by specifying a different name using the **-I *instance_name*** option.

For example:

```
extract -I name
```

forces EpiChannel to read all Registry values from **HKEY_LOCAL_MACHINE\...\Epiphany\Instances*instance_name***.

Other EpiChannel command examples follow:

- A command (in a system call) in which all Registry variables are set:
extract -j *job1*
- A command example in which no Registry variables are set:
\$\$EXC_CMD

Output Files

Jobs may log to files or databases, and these logs can be directed to multiple destinations simultaneously. For example, one log could be placed on the local machine and another on a network server.

The execution of jobs is logged as a unit. Statistics are collected for job steps and for entire jobs. The location of the log is established by associating data stores with the *log* or *working directory* roles for a job, for which you will use the Job dialog box in EpiCenter Manager (see Figure 54, on page 190).

Extracting New Rows Only

An efficient extraction process pulls only data from the source system that has changed or been added since the previous extraction. The way that the extraction process recognizes this subset of data depends on how the source system identifies changes in data.

Epiphany supports the following methods of identifying changed data:

- Date or Date/Time fields in source rows
A source row, or a table it joins to, contains a date or date/time indication of when it was inserted or last updated.
- Timestamp fields in source rows (Microsoft SQL Server only)
Values can be “stamped” in some way to indicate their status. Epiphany provides Microsoft SQL Server timestamp fields for source rows. A source row, or a table it joins to, has a column that contains Microsoft SQL Server data type *timestamp*. The database engine automatically updates this column whenever the row is inserted or updated. This field does not actually contain any record of the time that the update occurred, but rather values that always increment, such that any row edited after another row will have a higher timestamp.
- Highest numeric or string value in named columns in source rows
A particular column in a particular table is presumed to contain a value that is always increasing from row to row; for example, Order Number or Policy ID. For this method to be useful as a subset selection criterion, however, either the rows must never be updated, or else the column must be changed by the source system whenever an update occurs.

How EpiChannel Identifies Data To Be Extracted

The following topics describe how EpiChannel identifies which subset of data to extract:

- How EpiChannel reads values

EpiChannel attempts to get a consistent set of dates/timestamps/column values so that the same single moment in time is captured by the WHERE clauses generated by the extraction set identification macros. The extraction program minimizes the chance for changes to occur to one set of values, and not to another one, by reading all sets one after another with no intermediate actions. While this gives highly consistent values, it is possible that a change may occur between the time EpiChannel reads the first table/column and when it reads the last one, especially if multiple databases are involved.

The values read are stored as the “last” values only if the job commits its work. If any system call or SQL statement has an error that aborts the job, it is as if the value stamps had never been read or modified. In the unlikely event that a job aborts during the commit phase, the timestamps might not be synchronized with each other.

The date and timestamp values are read from a table-independent current value such as SYSDATE or TO_EPIDATE.

- Extraction subset extraction

Epiphany supports *extraction subset extraction* by reading and recording the current values of these fields at the start of the run and remembering these values for the next run. The values are made available in SQL through the SQL macros described in Appendix A, “Epiphany Macros.”

- Ignore Timestamps option

The Ignore Timestamps option, which you can set using EpiCenter Manager’s Job dialog box, causes all of the value range expressions to become (1=1), or true, which means that all data is selected without regard to its value range.

- “Priming” value range memory

The value range memory needs to be “primed” by a trial run whenever you add a new range expression. Basically, the first run causes the memory to start tracking the values for the next run, but all ranges are 1=1 for the current run. The second run causes the “current” values to be read, but all ranges are 1=1 because there are no last values. The third and successive runs have both current and last values, and the range expressions are valid. If you have changed the SQL and are in doubt, make a trial run, which will prime the value memory with any new value expressions.

- Removing and Re-adding Extraction Set Identification Macros

The presence of an extraction set identification macro in an enabled SQL statement causes the date, timestamp, or column values referenced in that expression to be read at the start of the extraction job and “memorized” if the job is successful. It does not matter if the macro is in a SQL comment, or even within another EpiPhany macro such as \$\$ORACLE[]. The macro is processed (and the values read) without awareness as to whether the proper parsing of the greater SQL statement indicates the expression is enabled.

Once a value is memorized, it stays memorized. If you remove the extraction set identification macro for some runs and later re-add it, the values memorized when the expression was last used will be the “start” range of the re-added macros. If you want to “forget” the previous values, open the extractor’s dialog box in EpiCenter Manager, select the Filters tab, and clear or delete the memorized value.

Memorized values are shared between expressions within an extractor. For example, if two extraction set identification macros reference the value of *current_orders.order_number*, then that value will be read and memorized just once. If one of these macros is removed, the other continues to read and increment the maximum *order_number*. If the removed macro is then re-added, it will expand with the incremented *order_number* value. If all extraction set identification macros that reference *current_orders.order_number* are removed, then the lookup and memorization of *order_number* will not occur. The next macro to reference that *order_number* will use the last value

Job Output

You may display EpiChannel output for debugging purposes on a screen or direct it to a file. The text is displayed on the screen in fixed fonts, such as Courier. The number of characters that can be displayed per line is determined by the number of columns. (You may use EpiCenter Manager's Job dialog box to set the job log width to 80 or 132.) The output is designed to be adequately displayed on low-end monitors.

At the top of the debugging screen/log, you will find the names and directory locations of the job's EpiMeta database, as well as its *job log*, *log file*, and *log database*.

Typically, the job begins with the truncation (deletion) of old staging tables and the clearing of indexes. The processing of each step is displayed sequentially; for example:

1.1 Processing step 'PreMfg' as an extractor

followed by the extractor's source and destination databases. Steps that are not enabled are numbered, but not shown.

Each operation follows on a separate line. You can identify the type of operation by the letters or symbol(s) that precedes it:

- **St** = A set of SQL statements. The members of this group can be either stand-alone SQL, extraction statements (pull/push), or semantic instances (data merges). Steps may be one of these types:
 - **< =** Stand-alone SQL applied to the source data store.
 - **> =** Stand-alone SQL applied to the destination data store.
 - **<> =** Pull/push SQL that transfers data from the source data store to the output data store.
- **SI** = Semantic instances.
- **Sy** = System calls.

Job Output

The EpiChannel output provides summary information that is based on the statement type. The summary information is prioritized and displayed in columns to the right of the operation. If there is not enough space, only the highest priority information is given.

For SQL pull/push statements, the output includes the time taken for the pull/push, the number of rows in the table before the operation, the number of rows transferred, and the number of rows in the table at the end of the procedure. If the log file's width is 132, the number of bytes is also shown.

Stand-alone SQL statements have time as their only metric. Semantic instances define their own metrics.

To conserve space, the summary information is labeled by two-digit codes (tokens) that appear to the right of their numeric values. These tokens are defined as follows:

- **Ti** = Time taken, rounded to the nearest second.
- **St** = Rows present in the table at the start.
- **Ex** = Rows extracted.
- **En** = Rows present in the table at the end.
- **UN** = Rows unjoined. These fact table rows contained references to dimension values not present in the EpiMart or in the newly extracted data.
- **PR** = Rows processed; a count of the gross number of rows examined.
- **IN** = Rows inserted; a count of the rows that were successfully moved into the EpiMart. Rows are sometimes omitted because they are earlier than previously recorded data, or because they are redundant to data present elsewhere in the same staging table.
- **MO** = Rows modified. Modifications typically adjust references to missing dimension values such that they become references to the predefined UNKNOWN dimension value.

For example, the lines:

```
Statement NameTime Metric1 Metric2 Metric3
```

```
St wb bump the test day
```

```
< wb bump the test day
```

```
<> appl real          0 St   3 Ex   3 En
```

```
SI Product   1 TI   3 IN   0 MO   3 PR
```

indicate the following:

- A set of statements called `wb bump the test day` was called.
- There is no summary information for this set.
- An execute-only SQL statement called `wb bump the test day` was run against the source system and took less than a half second to execute, and therefore has no time statistic.
- An extraction statement called `appl real` inserted 3 rows into a previously empty table and took less than half a second to execute.
- A semantic instance called `Product` took about a second to execute and processed 3 rows, modifying the contents of none, and inserting 3 rows as data into the EpiMart.

The summary table at the end of the report shows totals for the amount of time each operation took and the number of rows, bytes, and metadata objects that were extracted. Using this information, you can quickly determine if a problem resides in EpiChannel, the semantic instances, or in the extraction SQL statements or system calls.

Error Messages

An EpiChannel error message distinguishes between externally generated errors, such as database errors, and internally created messages, such as a file data store that references a nonexistent directory. Each error includes information about the database/file that caused the error, and which specific part of the code detected the error. Most errors contain information about the operations affected or aborted by the error, the metadata objects that specified these operations, and the program's plan of revised actions. Many errors also suggest corrective action.

There are specific error messages for the following:

- SELECT statements that do not match columns in the metadata. These display mappings indicate which return values were or were not accepted.
- Rows that fail to insert have their complete set of data displayed.

EpiChannel Debugging Levels

EpiChannel's verbose (**-v**) option allows you to select how much of the SQL displays on the screen during extraction. For a description of each debugging level, run EpiChannel with the **-h** or **-?** parameter. The default level shows jobs, steps, SQL statements, semantic instances, and statistics.

The lowest level shows only jobs and errors. Increasing levels show the SQL pre- and post-expansion of macros, templates, and blocks within semantic instances, or even each individual row returned from a SELECT statement. At the highest levels of verbosity, execution pauses between SQL statements and between returned rows, which allows you to trace behavior precisely.

You may even use EpiChannel interactively to alter the data fetched by a SELECT statement before the row is forwarded for insertion. See "Setting Breakpoints" on page 79.

Note: To turn off verbosity, select **-4**.

Setting Breakpoints

You may set breakpoints on SQL statements that change the debug level *before* the SQL is issued. These breaks can also be associated with particular rows in the return set. For example, it is possible to set a breakpoint just before row 3021 of the *Orders table extraction* SQL statement.

You can use the EpiCenter Manager's SQL Statement dialog box to set breakpoints (see Figure 52, on page 185). Follow these steps:

- Step 1:** Select the debug level. These levels correspond to the verbosity levels on the **extract** command line (or in the Execute Job dialog box shown in Figure 55, on page 192).
- Step 2:** Select the row number at which the debug level you selected above will go into effect; this is the row number immediately *before* the SQL statement is executed. If the row number is not empty, and the level is “row pause” or higher, the pull/push loop stops fetching before inserting that particular row and waits for permission to proceed. You may alter the data during this pause.

When you set a breakpoint, it stays set for the remainder of the job. This prevents you from having to set a debug level on entire groups, but it does necessitate your setting the breakpoint back to the original value later in the job flow if you do not want the entire job debugged (and remembering to remove this breakpoint).

All SQL issued against the source, destination, or metadata databases will be logged, if specified by the debug levels.

Trial Runs

You can run EpiChannel in trial-run mode, in which SQL is not actually issued against the source or destination systems. This option is recommended for testing the structure of an extraction job.

EpiChannel Output

Maximum limits can be set on the number of rows to be fetched from any SELECT statement. When used, these limits essentially cause the SQL to be checked for syntax on a small selection set. The execution time for all SQL executed is tracked and logged.

EPICHANNEL OUTPUT

You must provide EpiChannel with one and only one file data store with the role of *working directory*. A working directory is a log directory that is also used as the location for temporary files associated with the job.

You may, however, have any number of file or database data stores that have the role of *log*. The locations referenced by data stores with these roles are known as log directories and log databases, depending on their data store type. You may use more than one log database and more than one log file. In this case, all logging activity is duplicated to all of the listed data stores. Having a backup of each kind of data store is recommended if you require logging information to be constantly available for the administration of your system, or if data stores could possibly go off-line or be destroyed.

You may also use multiple listings so that one log is local to the host and one is remote. For example, during installation and early testing, you might direct one database log to a database hosted by Epiphany for the monitoring of your site's activities.

Note: The EpiCenter Manager default data stores may be sufficient for your site.

Log and Working Directories

The location specified for a *log* or *working directory* is a parent directory under which a new subdirectory will be created for every job. The subdirectories are named based on a timestamp from the start of the EpiChannel run. If there is already a subdirectory of that name, then a suffix is added to make the name unique to that directory. Under the subdirectory is a file called *jobname.log* with the log and informational messages from that job.

Log Files versus Log Databases

EpiChannel can log to both files and databases. Log files provide detailed information regarding a single job and log databases provide complementary but less detailed status and performance information about all jobs. Log files help you to determine the success of a job and suggest corrective action. Log databases enable you to analyze the behavior of jobs over time in order to detect troublesome areas, or areas of degrading performance.

Note: A log database is not a log device. A log device is a construct in SQL Server that logs database transactional activity.

MONITORING JOBS

You may use the Windows NT Performance Monitor to determine the current status of the database extraction process; for example, you can find out if the cause of a longer than normal extraction process is the result of network performance problems, or simply more rows being extracted and processed.

You may use one or more of these measures to monitor job behavior:

- The total number of jobs started. Use this to see transitional boundaries in the work being done.
- The total number of extraction nodes started. Use this to see transitional boundaries in the work being done.
- The total number of rows transferred in a pull/push operation. This value is reset with each extraction statement.
- The total number of bytes transferred in a pull/push operation. This value is reset with each extraction statement.
- The number of rows committed. This value is reset with each extraction statement.
- The rate of row transfer in a pull/push operation (rows per second).
- The rate of byte transfer in a pull/push operation (bytes per second).

Running the Performance Monitor

- Total number of semantic instance blocks called. Use this to see transitional boundaries in the work being done.
- The rate at which semantic instance blocks are called. This shows the percentage of the clock time spent in semantic instance calls.
- The number of meta objects in the job, which is a measure of the complexity of the workflow.

You may combine these metrics with the other measures available through the Performance Monitor. For example, the bytes per second can be displayed along with the TCP packets per second, the average length of the disk queue, and SQL Server-specific measures.

Running the Performance Monitor

See the *Epiphany Installation Guide* for instructions on how to set up the Performance Monitor for EpiChannel usage.

After setting up the Performance Monitor, you can monitor any EpiChannel jobs that are run on this machine. Follow these steps:

- Step 1:** Start the NT Performance Monitor. (It is located in the *Start\ Programs\Administrative Tools* menu on most machines.)
- Step 2:** Click the ADD or plus (+) button in the Performance Monitors' window.
- Step 3:** Select Epiphany Channels from the object list.
- Step 4:** Select the measures you want to monitor from the counter list.

MIRRORING: A AND B TABLES

The Epiphany system uses the metadata definitions of facts and dimensions to construct the physical EpiMart tables. Table naming conventions indicate how the table is used. For instance, if a base dimension table named *Customer* has been defined via EpiCenter Manager, the system constructs these tables:

- *Customer_0_A*
- *Customer_0_B*
- *CustomerMap_A*
- *CustomerMap_B*
- *CustomerStage*

The *_0*, *Map*, or *Stage* suffixes are needed for the normal operation of EpiMart. Note that the *Customer_0* and *Map* tables appear twice in this list: once each with a suffix of A and B. The purpose of this table “mirroring” is to allow extractions to occur without disrupting the live usage of a system.

At any time, either the letter A or B is “active,” meaning that all end-user queries will operate against tables with that suffix. This setting is determined by an option in the EpiCenter Manager’s Configuration dialog box (described in Appendix B, “EpiCenter Configuration.”) called *current_datamart*. If this option is set to A, then all end-user queries will be routed against tables that end in A (and all of the B tables will lie dormant). The B tables contain data that is one extraction older than the A tables.

During an extraction, the data in the active tables (A in this example) is never changed. Instead, the B tables are constructed by combining the A table data with any newly extracted data from the staging tables. In other words, the dormant set of tables is rebuilt every extraction with the latest data. Not only does extraction of the base tables (those that end with *_0*) occur in the dormant set, but aggregation is also performed on the B tables. In this way, end users can continue to query against the A tables, without knowing that extraction and aggregation are occurring simultaneously against the B tables.

SQL Limits for Facts

If extraction finishes successfully, then the value of the configuration setting *current_datamart* is toggled to be the dormant letter (B in this case). As soon as the Epiphany Application Server (described in Chapter 4) is notified that the data extraction was successful, all end-user queries will be directed against the B tables. In this way, the A tables, which contain one less extraction's worth of data, become the dormant set. The next time EpiChannel is run, the A tables will be recreated, and the letter A will once again become active.

Note the following:

- Aggregate building is always performed against the set of tables not equal to the *current_datamart* setting.
- In order to completely re-extract an entire system, the set of tables whose suffixes equal the *current_datamart* setting must be truncated. (This, however, affects current end users of the system and will be resolved by the introduction of a new semantic type that allows complete re-extraction without truncating any EpiMart tables.)

SQL LIMITS FOR FACTS

When configuring your datamart, observe these SQL limits, as detailed in the SQL Server documentation:

- MONEY facts are limited to +/- 922,337,203,685,477.5807
- QUANTITY facts are limited to +/- 999,999,999.99
- INT facts are limited to +/- 2,147,483,647

The selection of a datatype that is insufficiently large enough to store your data may affect the accuracy of Epiphany software extractions and queries.

CHAPTER 3

EpiCENTER MANAGER

EpiCenter Manager is Epiphany's versatile program for configuring, administering, re-configuring, and updating your database. You will use its graphical user interface to define your EpiMeta database table schema, configure ticksheets for end users, and set up the jobs needed for data extraction. You will also use EpiCenter Manager to perform administrative tasks, such as restricting access to ticksheets and sending e-mail notification of job status. When your schema changes incrementally, EpiCenter Manager allows you to update it "on the fly," without rebuilding the entire EpiMart.

This chapter describes how to use EpiCenter Manager to set up your databases for optimal extraction of your source data, to define jobs, and to administer your system. Instructions are also given for configuring Clarity, Relevance, and Momentum ticksheets.

PLANNING YOUR EPICENTERS

EpiCenter Manager is a program for experienced database administrators who are familiar with their site's database and database needs. Because incorrectly updating schema can corrupt an existing database, it is important that you understand Epiphany system concepts and how to use EpiCenter Manager in a way that is appropriate for your site. Before using EpiCenter Manager, be sure to read Chapters 1 and 2 of this *Guide*, which describe the Epiphany system and the database extraction process. Many of the key terms, such as EpiCenter, EpiMeta, and EpiMart are described in these chapters.

Before you configure an EpiCenter, your site needs to analyze its business model to determine:

- How many EpiCenters are appropriate.
An EpiCenter is a construct for organizing your metadata into a database called EpiMeta that defines a single EpiMart database. EpiMart is where the data extracted from your source systems resides. If your system has adequate memory and storage (see the *Epiphany Installation Guide*), and your business model warrants it, you may set up multiple EpiCenters. For example, you may have one for development and one for production.
- The number of constellations within each EpiCenter.
This includes the fact tables and dimension tables (and aggregates) that should be logically organized into a constellation. Although every EpiCenter has at least one constellation, most EpiCenters will have multiple constellations. The number depends on the site's business model; for example, one for each business process.
There is a separate constellation especially for the Momentum application. Momentum also has associated constellations, called Momentum Adjacent constellations. When you are working with Momentum, you will more than likely create adjacent constellations specifically for Momentum demographic queries.
- The kinds of Clarity, Momentum, and Relevance ticksheets you will configure for end users.

- The extraction process for your site.
Analyze the kinds of jobs and job steps needed; for example, you may want to create job flow chart(s). After you design the system at this higher level, you will be better able to—
 - Identify the extractors (SQL statements and semantic instances) needed to extract information from the source system(s) and write these extractors.
 - Define external tables.
 - Identify requisite system-call commands.

Note: For security purposes, two functions of EpiCenter Manager are available as separate programs: Web Builder (for creating ticksheets) and Security Manager (for assigning access rights to log onto the system and to access, modify, and save ticksheets).

GETTING STARTED

Follow these steps to start EpiCenter Manager and to register a database server:

Step 1: Complete the installation as described in the *Epiphany Installation Guide*.

The installation procedure includes creating new, empty databases on the server for the EpiMeta and EpiMart databases and installing the Epiphany software. Although you create two databases, EpiMeta is the one you work with using EpiCenter Manager. EpiMeta defines the database tables for your data warehouse (the EpiMart).

Step 2: Start EpiCenter Manager.

During installation this program executable (**EpiMgr.exe**) is placed in the Epiphany directory by default (**C:\Program Files\Epiphany**) and can be selected from the Start menu: *Programs\Epiphany\instance_name\EpiCenter Manager*.

The EpiCenter Enterprise Manager window is displayed.

Getting Started

At the top level is the Servers icon. Follow these steps to register your server:

Step 1: Choose Register from the Server menu. The Server Properties dialog box (Figure 9, on page 88) is displayed in the window.

Step 2: Select the Server Type from the drop-down list and enter the database server's name, and the username and password for this server.

Your database server machine icon and the EpiCenters folder (or directory) appear in the hierarchical tree structure (see Figure 10, on page 89). This server icon represents the server where the EpiMart and EpiMeta databases for this EpiCenter reside.

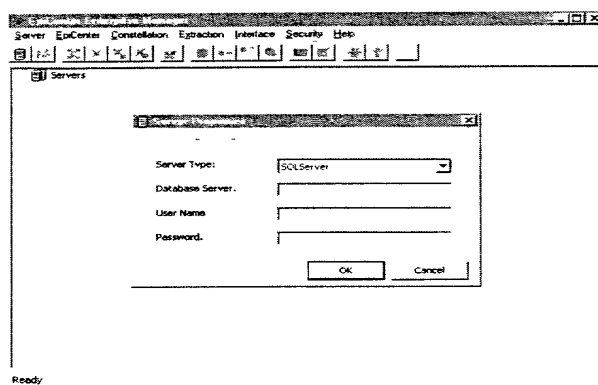


Figure 9: Registering the Database Server

Step 3: Click Cancel to close the empty Choose EpiCenter dialog box.

Later you may use this dialog box to select and open an already initialized EpiCenter as described in “Working with an Existing EpiCenter” on page 92.

When you register a server for the first time, after connecting to that machine, EpiCenter Manager displays the Register EpiCenter dialog box. Close this dialog to continue with the initialization.

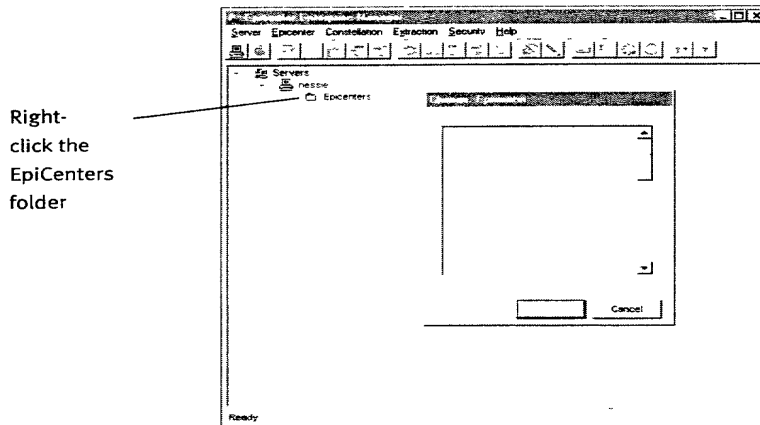


Figure 10: EpiCenter Enterprise Manager Window

Follow these steps to initialize your EpiCenter:

- Step 1:** Right-click the EpiCenters folder in the EpiCenter Manager Enterprise window (see Figure 10) and select Initialize EpiCenter from the pop-up menu. The Initialize EpiCenter dialog box is displayed (see Figure 11, on page 90).
- Step 2:** In the EpiMeta Name drop-down list, select the name of the new, empty database for the metadata that you created during installation (as described in the *Epiphany Installation Guide*).
- Step 3:** In the EpiMart Name drop-down list, select the name of the new, empty database for your customer data that you created during installation.

Getting Started

- Step 4:** The basic initialization is for Clarity only. Select Include Relevance and Include Momentum if you have installed these products.
- Step 5:** Enter the name of the directory for the default job log file. The default location is the Windows **TEMP** file location on the current machine.
- Step 6:** Click Build to start building a generic EpiMeta database (as specified in the *Epiphany Installation Guide*). The status of the build is displayed in the lower part of the (now expanded) Initialize EpiCenter dialog box.

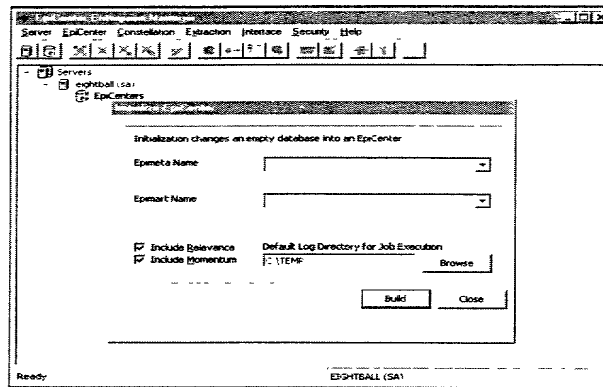


Figure 11: Initializing Your EpiCenter

The new EpiCenter icon is added to the directory tree with folders for Base Dimensions, Constellations, Extraction, Shared Interface, and Security (see Figure 12). Note that the icons in the window are arranged in an Explorer-like hierarchical tree.



The date dimension table for a new EpiCenter needs to be populated with dates appropriate for your installation. You can do this after you create your EpiCenter when you generate the schema as described in “Generating Schema” on page 214.

Note: Quarters 4-4.5 represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.

Working with an Existing EpiCenter

For descriptions of these date fields, see “General Settings” on page 285 in and Appendix D, “Physical Type Values”.

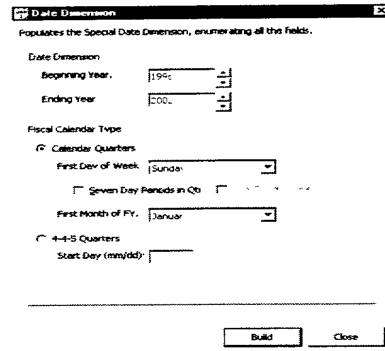


Figure 13: Date Dimension Dialog Box

For a discussion of how the Epiphany system treats time, see “The Uniform Treatment of Time” on page 31.

WORKING WITH AN EXISTING EPICENTER

If an EpiCenter has already been initialized, you may register it as follows:

- Step 1:** In the EpiCenter Enterprise Manager window, right-click the EpiCenters folder icon and select Register from the pop-up menu.
- Step 2:** In the Choose EpiCenter dialog box, select the EpiMeta for the EpiCenter.

The EpiCenter’s hierarchical tree is displayed in the EpiCenter Manager Enterprise window. You may modify this EpiCenter as appropriate for your site by following the instructions given in this chapter for those items you wish to change.

You can unregister an EpiCenter by right-clicking the EpiCenters folder icon (see Figure 10, on page 89) and selecting Unregister from the pop-up menu. The EpiCenter's hierarchical tree is removed from the window.

THE EPICENTER MANAGER WINDOW

Similar to other Windows applications, you may invoke EpiCenter Manager commands by using the main menu, toolbar icons, or by right-clicking icons in the window and selecting commands from a pop-up menu. Before you start configuring your EpiCenter, familiarize yourself with the main menu commands and the toolbar icons. The main menu commands are organized by the main EpiCenter Manager functions: Server, EpiCenter, Constellation, Extraction, Interface, Security, and Help. These menus are contextual and their commands are accessible when you are working within their function. For example, the Constellation commands are available when you working with a Constellation, but may be unavailable in other contexts.

EpiCenter toolbar icons provide access to special functions such as truncating tables, or displaying dialog boxes you use on a regular basis, such as the new Job dialog box and dialog boxes for defining facts, dimensions, external tables, and users and groups. A toolbar icon must be activated (highlighted) for you to use it. To activate a toolbar icon, select a related icon in the directory tree.

WORKING WITH EPICENTER MANAGER

The EpiCenter Manager window is organized in a hierarchical tree structure in which you right-click a plus or minus sign to expand or collapse a directory, as in Windows Explorer. In general, you will work down the tree structure in the window, right-clicking folders or icons to display a contextual pop-up menu, or you may prefer to use the equivalent main menu or toolbar commands. The instructions in this chapter use both ways interchangeably.

Working with EpiCenter Manager

Double-clicking an empty folder icon opens a dialog box in which you can define a new item, such as a new constellation. If this item has already been created, double-clicking expands or collapses the directory tree. Double-clicking an existing icon, such as the Default Job icon, opens its dialog box.

The following applies to all of your work with EpiCenter Manager:

- **Descriptions in dialog boxes**
The Epiphany system does not utilize the descriptions in the Description text area of the EpiCenter Manager dialog boxes. This description is for your documentation purposes only.
- **Updating and refreshing**
You may click the Update button in a dialog box to change the EpiMeta description of an existing row of data in the database “on the fly.”
If multiple users are running EpiCenter Manager, metadata changes made by one user are visible to others only after that user issues the Refresh command.
- **Unregistering and deleting items**
You can unregister servers and EpiCenters by right-clicking their icons and selecting Unregister from the pop-up menu. You can delete an item in the EpiCenter Enterprise Manager window, such as a base dimension table or constellation, by right-clicking its icon and selecting Delete, or by selecting it and pressing the Delete key.
- **Duplication**
The duplication feature allows entire objects to be copied or duplicated. (It is available by right-click in the main-menu, or by pressing Ctrl+C.) You are prompted to enter a name for the new object, which results in a complete “deep” (including all children) copy. This feature enables you to clone a base dimension table, fact table, extractor, and so forth.

EPICENTER MANAGER REPORT

The EpiCenter Report provides a record of all or part of your EpiCenter Manager configuration in an HTML file that displays in your browser window. (Figure 14 shows a sample report.) You can generate this report whenever you need a snapshot of your current EpiCenter (and use your Web browser to open saved reports for viewing). You may want to generate this report while you are developing your EpiCenter in order to record your progress.

To generate this report, either select one of your EpiCenters in the directory tree and choose Report from the EpiCenter menu, or right-click the EpiCenter folder and select Report from the pop-up menu. The Report dialog box (Figure 15) is displayed.

In the Report tab, enter the Report File name preceded by the directory location (If you do not enter this now, you will be requested to do so later.) If the Report File name is the same as another file, that file is overwritten.

Select all or a subset of the report topics. These topics reflect the major folders of the EpiCenter: Extraction, Interface, Security, Ticksheets, Schema, and Constellations. The Schema represents higher-level metadata that all constellations in the EpiCenter share.

You can display specific sub-topics; for example, selecting only Ticksheets results in a report that shows the attributes, measures usage, and filters for the EpiCenter. Optionally, you could select measure usage only as the report topic.

By default, the report is displayed after it is generated, and topics within the report are hyperlinked.

Click the Results tab to view the file location and status of the report's generation. The time that elapsed for a successful build is also given.

EpiCenter Manager Report

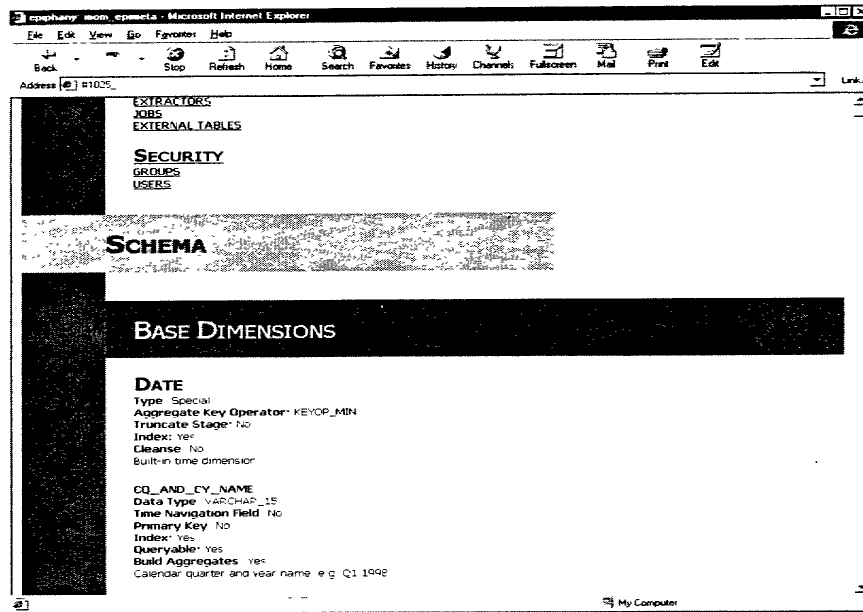


Figure 14: EpiCenter Manager Report

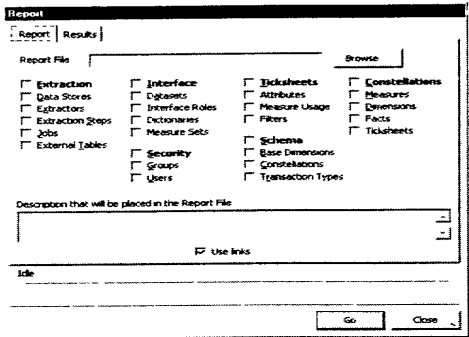


Figure 15: Report Dialog Box: Report Tab

SETTING UP AN EPICENTER

The purpose of setting up an EpiCenter is to define your site’s metadata, or schema. After defining it, you will use the Generate Schema command in the EpiCenter menu to write these definitions to the EpiMart database. The EpiMeta database points to your EpiMart (where your extracted data is stored).

The instructions in the rest of this chapter describe how to set up a single EpiCenter with one constellation. Some EpiCenters, however, will have more than one constellation. To create additional ones, simply repeat the procedure you used to create your first constellation.

Configuration

The Configuration dialog box provides a record of the current configuration of your EpiCenter. It shows general configuration settings, lists transaction types, defines how measure units are displayed, shows the labels for display option names as they appear on ticksheets, and describes the ticksheet types. (Choose Configuration from the EpiCenter menu to view these settings.)

09625518.072500

Base Dimension Tables

Other than entering your mail password in the General tab, the information in this dialog box should need little, if any modification. (Most of these settings can be set using other EpiCenter Manager dialog boxes, or are entered automatically as you configure the EpiCenter.) See Appendix B, “EpiCenter Configuration.” for more information.

Base Dimension Tables

The base dimension tables are physical dimension tables that can be used multiple times within a constellation or across constellations (depending on the dimension roles associated with a fact table). A base dimension table consists of dimension columns. These are columns in a physical base dimension table that hold attributes extracted from the source system.

All EpiCenters are created with two default base dimension tables: Date and Transtype (transaction type); therefore, the date and transtype dimensions occur in every fact table. (Remember that conceptually the star schema has the fact table in the middle and the dimension tables radiate outward as points of the star.) The use of a single date dimension throughout the system ensures a consistent treatment of time.

Transaction type dimensions are necessary because the EpiCenter must be able to distinguish among different rows in a single fact table, such as shipping transactions versus bookings, and bookings versus booked returns. For instance, an Order fact table might contain both a BOOK and SHIP transaction type, differentiated by the value of the column *transtype_key*. The *transtype_key* points to the Transtype base dimension table. The transtype may be viewed as a “slice” of the fact table; semantically, the transtype slice acts as a separate fact table.

You will use the Base Dimension dialog box to—

- define the EpiCenter’s base dimension tables, such as Product, CostCenter, Account, Subaccount, or Project.
- configure the dimension aggregates used for aggregate building.

A typical star schema has multiple dimensions associated with a fact table. For example, an Order fact table of 20 million rows may have the dimensions Customer (1000 rows), Product (3000 rows), and Date (365 rows). In theory, you could construct a star schema that has only one dimension; that is, a supra-dimension that contains all three dimensions. The resulting dimension table, however, would need one row for every possible combination of dimensionality in the fact table.

Why not build a single dimension table in this case? The Customer, Product, and Date data is independent of each other. Three smaller dimension tables with 1000, 3000, and 365 rows individually combined with the fact table can represent the same information as the number of distinct combinations that actually occur in the data: 5,000,000. The only trade-off (other than that 5,000,000 rows in a dimension table exceeds the system limit) is that each dimension has an associated dimension role key in the fact table that is a 4-byte integer, and these bytes accumulate. In this extreme example, the extra dimension role keys are worth the extra space.

Sometimes combining two small, independent dimensions may make sense. For example, SalesPerson has ten distinct values in the Order fact table, and Promotion has 20 distinct values. Even if these two dimensions are completely independent, building a composite dimension with several hundred rows, which is very small, can save a key in the fact table. These are considerations to keep in mind when defining your base dimension tables.

To define a new base dimension table:

Right-click the Base Dimensions folder and select New Base Dimension from the pop-up menu. The Base Dimension dialog box (Figure 16) is displayed.

Base Dimension Tables

This dialog box has three tabs: General, Column Sets, and Built Aggregates.

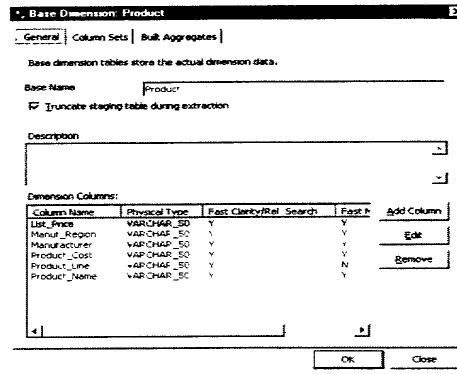


Figure 16: Base Dimension Dialog Box: General Tab

You can use the General tab to define the base dimension table:

- Step 1:** Enter the name of the base dimension table, such as *Product*. This is the name of the physical table in the database; underscore zero (0) will be added to this name to form the base table. The maximum length is 20 characters.
- Step 2:** By default, the data in the base dimension staging tables is deleted (truncated) after extraction.

In most cases, you will want to truncate staging tables. You may, however, want to de-select the *Truncate staging table during extraction* option when an execute-only SQL statement truncates the staging tables based on criteria you specify for the extraction process. For example, this SQL could delete all records whose foreign keys matched, or all records over seven-days-old.

Alternatively, you may use a single dialog box to define all tables within a constellation that will be truncated prior to extraction (see “Truncating Tables” on page 198).

- Step 3:** Click Add Column to enter one of the table's column names.
- Step 4:** In the Dimension Column dialog box, enter a name and a description for the column.
- Step 5:** Selecting the *Indexed for fast Clarity and Relevance search* option includes this dimension column with other selected dimension columns in a inverse (non-unique) index for fast lookup. De-selecting this option for selected dimension columns saves space, but increases search time.
- Step 6:** *Can be used in Momentum filter* must be selected in order for this column to be moved into the Momentum tables.

Figure 17: Dimension Column Dialog Box

- Step 7:** Select the physical type from the drop-down list. The physical type you select is replaced by its associated database type. The translation of physical type to database type depends on your RDBMS platform. See Appendix D, "Physical Type Values" for descriptions of these physical types.

Defining Dimension Aggregates

Step 8: Enter a default value for the column.

By default, each dimension column assumes the literal value UNKNOWN for each of its textual attributes. Leave this field blank to accept the default. To override this value so that end-user queries will return a different value, enter a literal string (without quotes). Your string will be used instead of UNKNOWN. See “The UNKNOWN Dimension Row” on page 64.

DEFINING DIMENSION AGGREGATES

A *dimension aggregate* represents a dimension table in which one or more of the columns have been removed, and the rows have been collapsed onto one another to remove duplicates. (In the example given in “Aggregate Building” on page 55, if Customer Name is removed from the base Customer table, only two distinct Regions are found in the table, so only two rows are needed in the Region aggregate table.)

You can use the Columns Sets tab of the Base Dimension dialog box (Figure 18, on page 104) to define which aggregates will be built the next time that the Aggbuilder program (**agg.exe**) is executed.

Column sets are subsets of the full list of dimension columns for the associated dimension base table. The base dimension column sets determine which base dimension aggregates the system builds. The number of rows in the dimension aggregate equals the number of distinct column set values. For aggregates, each column set represents a *potential* aggregate to be built. The Aggbuilder program does not build all defined column sets, it builds only those columns sets that are included in fact aggregates (see “Aggregation and Aggregate Grouping” on page 115).

Often one of the columns “drives” the set because sets usually represent a natural hierarchy in the data. The other columns are higher level roll-ups.

When defining dimension column sets, keep the following in mind:

- Do not place two dimension columns with high-cardinality (many values), and whose data is independent of each other in the same set. For example, City and CustomerType both have thousands of distinct values and are unrelated to each other.
- The size of the dimension aggregates greatly influences the size of the fact aggregates.

To define dimension aggregates, follow these steps:

Note: The data you enter in the Column Sets tab is used by the Aggregate Building process the next time **agg.exe** is run. Your changes do *not* affect the current EpiMart tables.

Step 1: Enter the name of the column set in the text box, and click New.

This name is a logical identifier only, and is not used during the building of the Actual table in the EpiMart. Instead, EpiCenter Manager will append unique numbers after the base dimension table name.

Step 2: Select *aggregate* as the Set Type from the drop-down list. (Mini-dimensions are used by the Momentum application and are discussed in “Using Momentum” on page 160.)

Step 3: Select or de-select *Include in default aggregate groups*.

Selecting *Include in default* for this column set affects the behavior of the default aggregate group that exists in each constellation (see “The Default Aggregate Group” on page 117).

Step 4: Add column names for this set by clicking New. Select columns from the list of columns for this base dimension table.

Add additional (multiple) column sets by repeating the above steps. See the next section for more information.

Multiple Dimension Column Sets

Multiple Aggregate set types are used when a base dimension table with many columns can be aggregated in different ways. Generally, the fewer columns included within a column set, the smaller the resulting dimension aggregate, and the faster queries that use fact aggregates joined to that dimension aggregate will respond.

For instance, if a customer dimension has several fields related to customer geography (City, State, Zip, and so forth) and several other fields related to demographics (age, marital status, years of education), then EpiCenter Manager can choose two or more dimension column sets (which can overlap) for maximum coverage of queries. At query time, Epiphany's aggregate navigation mechanism will choose the smallest aggregate that satisfies the user's request. Thus it is beneficial at query time to have built many small, highly specialized aggregates. The penalty for building many dimension aggregates is that it takes longer to build aggregates, and requires more disk space.

Note: A small change in the number of dimension aggregates can cause a large change in the number of fact aggregates. The Aggbuilder program builds only those column sets included in fact aggregates (see "Aggregation and Aggregate Grouping" on page 115).

Dimension Aggregate Browsing

While configuring your Epiphany system, you may wish to browse the list of aggregates that Aggbuilder has built (according to the instructions in the Aggregate Group dialog box described in "Aggregation and Aggregate Grouping" on page 115). You can use the Built Aggregates tab of the Base Dimension dialog box to browse the aggregates for either the A or B set of tables.

09625548.072500

Setting Up a Constellation

EpiCenter Manager uses a mirroring structure in which all EpiMart tables are in the database twice, once for each of the _A and _B tables. Only one set of tables, however, is active at a given time. See “Mirroring: A and B Tables” on page 83 for more information.

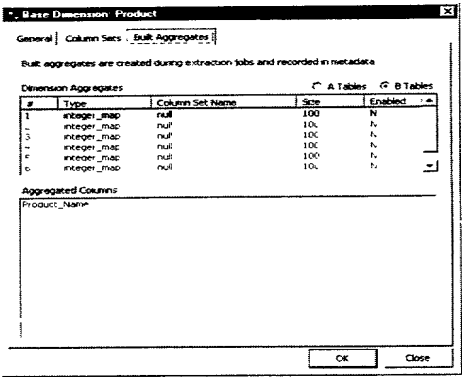


Figure 19: Base Dimension Dialog Box: Built Aggregates Tab

SETTING UP A CONSTELLATION

Within an EpiCenter, constellations serve to group fact tables that have similar dimensions. Instructions are given here for setting up a single constellation for Clarity and Relevance. To set up another constellation, repeat the entire series of steps in this section.

Note: For instructions on setting up a Momentum constellation, see “Using Momentum” on page 160.

To set up a constellation for Clarity and Relevance:

Step 1: Choose New Constellation from the EpiCenter menu.

Step 2: Enter the new constellation name and any description.

The New Constellation icon appears in the Constellation folder. It has these sub-folders Facts, Dimension Roles, Aggregates, Measures, and Ticksheets. Although you can usually proceed down the tree as you define your EpiCenter, you need to define dimensions and facts before aggregates.

Defining Dimension Roles

A base dimension table may have a superset of data that can be used for different purposes. For example, a Customer base dimension table may contain Bill-to and Ship-to data. Dimension roles enable base dimension tables to be used more than once in a constellation.

A dimension role indicates the single usage (such as Bill-to data only) of a base dimension table by all of the fact tables within a constellation. A row in a fact table may have several foreign keys that point to the same base dimension table, but the role usage may differ (for example, some relate to Bill-to and others to Ship-to data). The dimension role maps the fact foreign keys to the correct base dimension table.

To define roles for this dimension table:

- Step 1:** Right-click the Dimension Role folder and select New Dimension Role. The Dimension Role dialog box is displayed.
- Step 2:** Select the base dimension name from the drop-down list.
- Step 3:** Enter the Dimension Role Name and a description, and click OK. The dimension role icon is added to the tree.
- Step 4:** Repeat the above steps for any other base dimension tables that have multiple roles.

Every dimension role must have a parent base dimension table. If there is no base dimension for this role, click New in the Dimension Role dialog box. You can use the Base Dimension dialog box that displays to create a new base dimension.

Degenerate Dimensions

Degenerate Dimensions

Certain numbers in fact tables such as order, invoice, and bill of lading numbers have foreign keys with no corresponding dimension table. The only data of importance is the number itself, all of the other information of value has been extracted into other dimension tables. These numbers are called degenerate dimensions.

A degenerate dimension is a textual field stored directly in the fact table. It is part of a constellation, and all facts in the constellation inherit it. If you use degenerate dimensions, you need to inform the system of their existence. To define a degenerate dimension:

Note: Choose New Degenerate Dimension from the Constellation menu. Enter the degenerate dimension name and a description in the Degenerate Dimension dialog box.

Defining Fact Tables

A fact table is a physical table that holds numeric data in its columns. It also represents the intersection of a series of dimension keys. A fact table consists of dimension role foreign keys, degenerate dimension keys, and fact columns.

As mentioned, the transtype (transaction type) dimension occurs in every fact table. The transtype is a “slice” of the fact table that functions as a separate fact table. You can configure one fact table with multiple transtype base dimensions, or configure multiple fact tables. The advantage of having multiple transtypes is that measures that require two transaction types, such as BOOK and BOOK_RETURN need to issue only one query. The advantages of multiple fact tables are that queries that need only one transtype have fewer rows to navigate, and that aggregate tables are smaller.

To define a fact table:

Step 1: Right-click the Facts folder icon, and select New Fact. The Fact Table dialog box (Figure 20) is displayed. This dialog box has tabs labeled General, Custom Index, and Built Aggregates.

If Momentum is installed, there is also a tab named Momentum, which applies only to fact tables in Momentum Adjacent constellations. You will use this tab when you set up a Momentum constellation as described in “Using Momentum” on page 160.

- Step 2:** In the General tab, enter the fact table’s name and text that describes this fact table. The maximum length is 20 characters.
- Step 3:** By default, the data in the fact staging tables is deleted (truncated) after extraction. In most cases, you will accept the default.

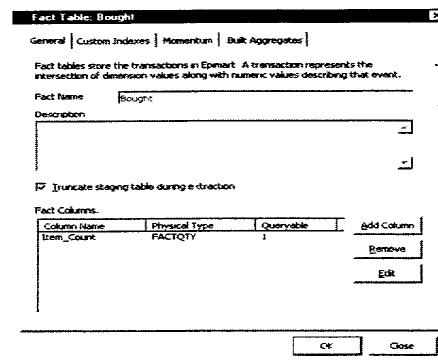


Figure 20: Fact Table Dialog Box: General Tab

Defining Fact Columns

Defining Fact Columns

Fact columns contain the actual customer numeric data; such as, *net_price* or *number_units*. You will use the General tab of the Fact Table dialog box to define a fact column:

Step 1: Click Add Column. The Fact Column dialog box (Figure 21) is displayed.

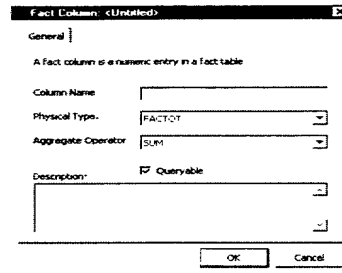


Figure 21: Fact Column Dialog Box

Step 2: Enter the column name and select the physical type from the drop-down list. The physical type you select is replaced by its associated database type. The translation of physical type to database type depends on your RDBMS platform. See Appendix D, “Physical Type Values” for descriptions of these physical types.

Note: For this release, the FACTMONEY, FACTQTY, and EPIINT physical types are supported for fact columns.

Step 3: Select the aggregate operator from the drop-down list. Aggregate operators determine how the Aggbuilder program calculates the facts. (For this release, SUM is the only option.)

Step 4: The Queryable option is selected by default. If this option is not selected, the column will not be moved into the Momentum tables.

Step 5: Enter a description, and click OK to return to the Fact Table dialog box. The new fact column appears in the listbox.

Step 6: Repeat the above steps to define another column.

To remove a column name, select it and click Remove.

Custom Fact Indexes

In general, aggregates satisfy high-level queries while indexes satisfy highly selective queries. Query drill-downs on a ticksheet transition from broad to selective. In terms of aggregates and indexing, the sequence of a drill-down transition from top to bottom might be—

Step 1: A small aggregate (for example, Business Unit equals Copiers).

Step 2: A larger aggregate (month equals January 1999).

Step 3: An index on a large aggregate (Customer Region equals West).

Note: Aggregate tables are indexed the same as their base tables on those columns that still exist in the aggregate. Aggbuilder does not build the same index twice simply because of missing columns.

Step 4: An index on a base dimension table (drill down by Customer Name equals John Doe).

A custom fact index is the metadata definition of indexes to build on fact tables in EpiMart. It is an ordered list of dimension roles. You can use EpiCenter Manager to configure custom indexes to dramatically improve query performance for selective queries. To actually build custom indexes, however, you must use the semantic template Custom Fact Index, which is described in Appendix F, “Semantic Types.”

By default, each fact table has a single clustered index with the leading term of *date_key* which allows fast filtering based on time. (The leading term of an index is the most important term; for example, a phone book is indexed by last name; you cannot locate someone by first name only.)

09625548 072500

Custom Fact Indexes

If end users typically apply selective filters to other dimension roles such as customer or product, then configuring custom indexes with these leading terms will improve system performance. (See “Custom Fact Indexing” on page 62 for a more in-depth discussion of this topic.)

You can use the Custom Indexes tab of the Fact Table dialog box (Figure 22) to create a new custom index:

Step 1: In the Indexed Columns panel, click New.

Step 2: Select one or more dimension roles from the Choose dialog box to add to this index. The first dimension role will be the leading term.

To delete the index, select it and click Remove.

You can use the Up and Down buttons in the Dimension Roles panel of the dialog box to change the order of the dimension roles in the index.

To modify dimension roles for an index: In the Dimension Roles panel, select the index and click Add Dim to add additional roles.

To remove dimension roles from an index, select the index in the upper panel. In the lower panel, select a dimension role and click Remove.

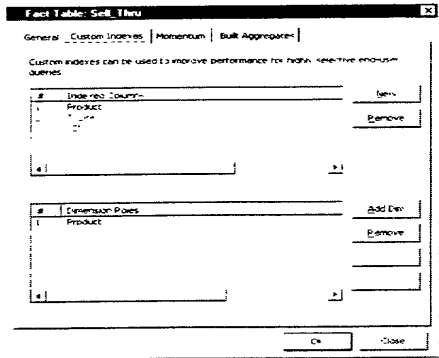


Figure 22: Fact Table Dialog Box: Custom Indexes Tab

Fact Aggregate Browsing

While configuring your Epiphany system, you may wish to browse the list of built fact aggregates to determine which tables are actually available for the query machinery. (See “Aggregation and Aggregate Grouping” on page 115 for more information.)

You can use the Built Aggregates tab of the Fact Table dialog box (Figure 23) for this purpose. As with dimension aggregate browsing, you can browse either the A or B set of tables, as appropriate. Follow these steps:

Step 1: In the upper panel of the tab, select a dimension role from the Role listing and then select a filter from the drop-down list. (Make one selection per dimension role.)

Filtering specifies that a dimension role—

- should not be included in the aggregate. (Not Included)
- should be included as the base table. (Base)

09625518.072500

Fact Aggregate Browsing

- should be included as a specific dimension aggregate (corresponding to a dimension column set). (No Filter Applied)

Other filters are user definable in the Column Sets tab of the Base Dimension dialog box.

Step 2: Fact aggregates appear in the Fact Aggregates listbox.

Each fact aggregate has an associated number. Although this numbering is arbitrary, it may be helpful when debugging query logs since the logs use these numbers. The size and compression ratio (number of rows in this aggregate divided by the number of rows in the base fact table) are also given. You can click the Size or Compression headings to sort aggregates by size. This sorting shows which aggregates are most valuable (smaller compression ratios mean greater aggregation).

Clicking a row in this listbox displays the following in the Aggregate Contents panel of the tab: the dimension roles that are included for this fact aggregate, and the dimension aggregates (or the base table) that the fact aggregate joins to.

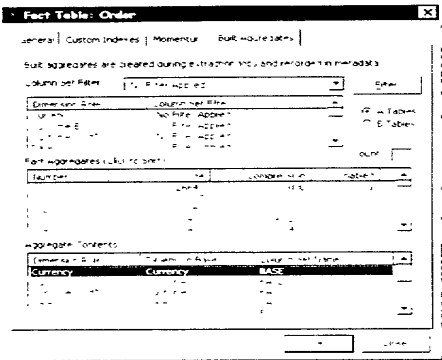


Figure 23: Fact Table Dialog Box: Built Aggregates Tab

Aggregation and Aggregate Grouping

Aggregates are facts that the system pre-calculates for a group of dimensions. (These dimensional groupings are determined by the groups you set up for a base dimension table.) An *aggregate group* is a set of instructions (a metadata definition) that tells Aggbuilder which aggregates to build on one or more fact tables. The fact aggregates that are actually built are determined by a combinatorial expansion based on the dimension role possibilities.

Unlike dimension aggregates, fact aggregates are not fully enumerated through the use of EpiCenter Manager. The reason for this is simple: while base dimensions typically have only a few aggregates, facts usually have many.

EpiCenter Manager simplifies aggregation by providing aggregate groups that you define using the Aggregate Group dialog box (Figure 24, on page 117 shows this dialog box for the Default Aggregate Group). For each aggregate group, you will specify the facts in the constellation that share it. (An aggregate group applies to a single constellation.) If two or more facts share an aggregate group, then the Aggbuilder program will follow the same set of instructions for each of the facts.

In general, when building aggregates, aim for a combination of broad fact aggregates (some compression/high query coverage) and narrow fact aggregates (high compression/low coverage).

To open an existing Aggregate Group dialog box, double-click it. (To open a new Aggregate Group dialog box, expand the Constellation tree and right-click the Aggregates folder. Select New Aggregate from the pop-up menu.)

You can use the General tab of the Aggregate Group dialog box to define an aggregate group:

- Step 1:** Enter an aggregate group name (and description).
- Step 2:** Select Enabled for Aggbuilder for the Aggbuilder program to use this grouping.
- Step 3:** Select the facts that share this aggregate. Click Add Fact and select facts from the Choose dialog box.

09625518 072500

Aggregation and Aggregate Grouping

Dimension roles in the aggregate group determine how fact aggregates are produced. The choices for inclusion or exclusion of a dimension role in an aggregate group are—

- **Exclude the Dimension Role**
Leave the dimension role out of the fact aggregate completely. If a dimension role is not included in the aggregate group, then *all* aggregates that result from this aggregate group do not include that dimension role.
- **All Columns in the Base Dimension**
Perform no aggregation on that dimension role.
- **Column Set**
Perform aggregation on the dimension columns that comprise the Column Set. When dimension aggregates are built during the aggregation process, each column set for a base dimension will become a dimension aggregate.

You can use the Definition tab of the Aggregate Group dialog box to add dimension roles for the aggregate group and to specify the ways to include them:

- Step 1:** In the upper panel, Click the Add Dim button.
- Step 2:** Select the dimension roles to be considered for these aggregates from the Choose dialog box.
- Step 3:** Select the ways to include the selected dimension role in fact aggregates by selecting a dimension role in the upper panel and clicking Add Set.
- Step 4:** Select the column sets for this dimension role.

The total number of Aggregates is shown on the tab.

The Default Aggregate Group

Each constellation has a default aggregate group that has special semantics to make the configuration of aggregates simpler. You cannot delete this group, although you may disable it.

The date dimension is automatically included in this group, since many end-user queries involve time. Date is included as an Exclude Dimension Role and Column Set. The Column Set is determined by which column sets of the Date Dimension dialog box are set to Default.

When a new dimension role is added to the constellation, the default aggregate group is automatically modified to include that dimension role. All Column Sets declared as Default for that base dimension table are included in the set of column set possibilities for the dimension role.

In all other respects, the Default Aggregate Group is identical to any other aggregate group.

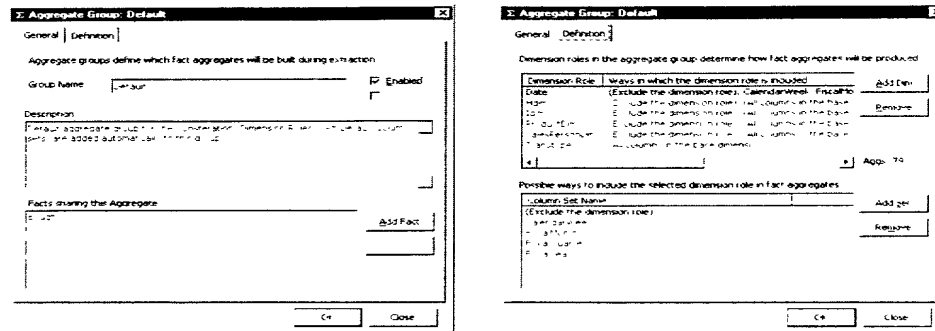


Figure 24: Default Aggregate Group Dialog Box

Measures

MEASURES

A *measure* is a business calculation that is the result of an arithmetic combination of fact columns. Each constellation has associated measures, which are organized alphabetically by name in the EpiCenter Manager's Measures folder.

When a user selects measure column *elements* in ticksheet columns, such as Average Sales Price, BookedOrders, and Gross, the calculations that apply to this combination of those elements depend on the measure to which they are mapped (for example, ASPBookedOrdersGross). These calculations are the values returned in the cells of Clarity and Relevance reports.

This section describes how to define measures and how to map them to column elements. *Measure mapping* is the means by which the Epiphany system knows which measure to apply when a user selects measure column elements from ticksheet columns. (Measure mapping does not apply to Relevance Influence ticksheets.)

Defining a Measure

To define measures for the constellation, use the Measure dialog box (Figure 25):

Step 1: Right-click the Measures folder and select New Measure from the pop-up menu. The Measure Builder dialog box (Figure 25) is displayed.

Note: Changes you enter in this dialog box are saved when you click OK, or simply close the dialog box. There is no Cancel button.

Defining a Measure

Example units of measurement are—

- Currency (for money units)
- Per cent (for percentages)
- Units (for the count of an item)

Measure Terms

You can use the Measure Terms panel of the Measure dialog box (see Figure 25) to define the steps that determine how the measure is calculated. A *measure term* is one component of an arithmetic expression that makes up a measure. A measure term refers to the aggregation of a single fact column in a fact table, such as *SUM(Order.net_price)* with a particular transaction type. Each term applies to a specific fact table, fact table column, transaction type, or backlog type (if applicable).

One measure term is combined with other measure terms to create a composite measure. The Epiphany system converts these steps to appropriate SQL SELECT statements. You will use Reverse Polish Notation to construct measure definitions. See “Reverse Polish Notation” on page 122 for more information.

To define a measure term step:

- Step 1:** Select one of the SQL operators from the drop-down list: SUM, COUNT, COUNT DISTINCT (or the negative values of these operators, such as -SUM or -COUNT DISTINCT).
- Step 2:** Select the fact table where the data resides, the fact column, transaction type, and backlog type.

The fact column is one of the columns of the fact table. All facts have an associated transaction type. Transaction types function as “slices” of a fact table; for example, all facts that are booked returns can be viewed as one slice of the fact table. Semantically, this slice functions as a separate fact table. For a description of the kinds of transaction types, see Appendix B, “EpiCenter Configuration.”

09625518.072500

Defining a Measure

The backlog types are BEGIN or END; or leave blank if the backlog type does not apply. You can use a backlog type when a measure term (a single line of the measure definition) should exhibit accumulating behavior. Normally, when running a report of Sales by Month, for example, the report shows only the sum of transactions that occurs in each month of the report. When a backlog type is used, however, the columns of the report show accumulated values from previous months, in addition to the current month. You can use BEGIN to show the accumulated value at the beginning of each period, and END to show the ending accumulated value.

For example, assume that report of sales by month transactions shows \$10 for June, \$20 for July, and \$40 for August. A report of the beginning backlog shows the accumulated value at the beginning of each month:

June	July	August	September
\$0	\$10	\$30	\$70

A report of the ending backlog shows the accumulated value at the end of the month:

June	July	August
\$10	\$30	\$70

- Step 3:** Click Add to add this as the first step in the Measure Terms panel.
- Step 4:** If appropriate, add another measure term by repeating Steps 1 through 3.
- Step 5:** Click the appropriate arithmetic operator to be applied to the measure terms: add (+), subtract (-), multiply (x), or divide (/). See “Reverse Polish Notation” on page 122 for examples.

Reverse Polish Notation

The bottom panel of the Measures dialog box (see Figure 25) displays a read-only infix display, or representation of the measure calculation, which translates the Reverse Polish Notation calculations to standard calculations. If the measure calculation does not currently make sense (for example, Term. Add. Add), the notation in the lower panel displays in red with one or more missing or extra term tags. You can save the measure, but it will not work in the Application Server.

Important: The removal of fact or dimension columns can remove measure terms, which invalidates their calculations. In this case, check all of the measures to determine if there are any infix displays in red.

To edit an existing measure term, right-click its Measure folder and select Edit from the pop-up menu, which displays its Measure dialog box. Modify this as described above. Click Update to show the changes in the dialog box, and click OK to save them.

To delete a measure, right-click its folder and select Delete from the pop-up menu. You can also use this pop-up menu to duplicate or export the measure.

Reverse Polish Notation

The Measure Terms panel in the Measure Builder dialog box uses Reverse Polish Notation³ (RPN) to construct measure definitions. RPN operations are performed in a last-in, first-out (LIFO) basis. All of the values to be operated upon are placed in a stack. Then the top two are operated upon and the result of that operation is placed in the stack, replacing the previous two values. Then the next top two are operated on and the result placed in the stack, and so forth.

For example, using Reverse Polish Notation for this calculation:

$1 + (2 * 3) = 1, 2, 3, *, +$ in RPN

means that 1, 2, and 3 are placed in the stack. The last two items (3 and 2) are multiplied, and the resulting value 6 is placed in the stack. The stack now holds 6 and 1, which are added, and the result 7 is placed in the stack.

³ Reverse Polish Notation is named for its Polish inventor, Jan Lukasiewicz.

In contrast, applying RPN to the calculation:

$(1 + 2) * 3 = 1, 2, +, 3, *$ in RPN

means that 1 and 2 are placed in the stack. These items are added, and the result 3 replaces them. Next, the value 3 is placed in the stack (the stack now holds 3 and 3). These items are multiplied and the result is 9.

Example 1 below shows how RPN is used to define a measure definition (in the Measure Builder dialog box). The Average Sales Price (ASP) for Booked/Gross Orders equals the total number of dollars received, divided by the total number of units shipped. This is represented as the following SQL SELECT statement:

```
SUM (Order.net_price)
SUM (Order.number_units)
div
```

Example 1

This SELECT statement is calculated using RPN as follows. The sum of all the Order net prices is calculated and placed in the stack. Then the sum of all the number of units is calculated and placed in the stack. Next, the division operator is applied to the top two items in the stack. It takes the next item in the stack, the total net price received, and divides it by the total number of units shipped, and the result equals the ASP.

For the operators that apply to the aggregates, use the arithmetic operators: add, sub, mult, and div.

Example 2 shows a similar calculation. This time the ASP is calculated for booked net orders (booked orders minus returns). Here the sums of two Order net prices are added (the returned items are represented as a negative number) and placed in the stack. Then the sums of two Order number of units (the returned items are represented as a negative number) are added and placed in the stack. The remainder of the calculation is the same as for Example 1 above.

Ticksheet Types

```
SUM (Order.net_price)BOOK
SUM (Order.net_price)BOOK_RETURN
add
SUM (Order.number_units)BOOK
SUM (Order.number_units)BOOK_RETURN
add
div
```

Example 2

TICKSHEET TYPES

A ticksheet is Epiphany's user-interface form that allows authorized users to construct queries. It is called a *ticksheet* because users select, or tick, items on a page that they open in their Web browser.

A description of the Clarity, Relevance, and Momentum ticksheet types follows.

Clarity

- Table & Charts.

A Clarity ticksheet produces reports for selected attributes (the columns and rows of the report) with measure values inside the cells. Corresponding bar and pie charts are also provided.

Relevance

- Best & Worst

Relevance Best & Worst searches thousands of Clarity tables automatically and builds a list of the most interesting results within those tables.

- **Influence**
Relevance Influence builds models of a company's data to help identify the factors with the greatest impact on the business.
- **Lifecycles**
Relevance Lifecycles predicts performance of new business entities, such as products, customers, sales representatives, and so forth based on the historical performance of related entities.
- **Profiling**
Relevance Profiling shows a page of thumbnail charts that help you quickly understand your data.
- **Quarter Projections**
Relevance Quarter Projections answer the question, "Given where I am today, and given the history of previous quarters, how is this quarter likely to end? Will I meet my goals?"
- **Trends**
Relevance Trends fits a smooth curve to time-varying data, and forecasts one or more periods into the future.

Momentum

Momentum generates lists using demographic data for individuals or groups. You can use Momentum to define and generate lists of people or things that match your requirements. For example, Momentum can provide a list of all customers with outstanding invoices over 90 days, or all customers who have purchased one set of products but not another. You can also use Momentum lists as filters in Clarity and Relevance.

The Momentum ticksheet types are Individual and Household.

CONFIGURING A CLARITY OR RELEVANCE TICKSHEET

Note: The Ticksheet Types tab of the Configuration dialog box (Figure 82, on page 294) enables you to disable an entire application by choosing the application from the drop-down list and de-selecting Enabled. For example, you may want to test Clarity ticksheets, but have not yet configured any Relevance or Momentum ticksheets.

To start building a new Clarity or Relevance ticksheet, right-click the Ticksheets folder and select New Ticksheet from the pop-up menu. The appropriate Ticksheet dialog box for the ticksheet type is displayed. A ticksheet dialog box has six tabs: General, Attributes, Filters, Measure Mappings, Files, and Copy Ticksheet Items. (Relevance Influence ticksheets have the Measure Sets tab in place of Measure Mappings; additional instructions for configuring Influence are given in “Relevance Influence Ticksheets” on page 152).

First, configure a new Clarity or Relevance ticksheet (other than a Relevance Influence ticksheet). Instructions are given below for completing each tab in the Ticksheet dialog box.

Note: The configuration of Relevance and Momentum ticksheets differs in some aspects. These differences are addressed in “Configuring Relevance Ticksheets” on page 146 and “Using Momentum” on page 160.

General Tab

- Step 1:** You can use the General tab to enter the name of the ticksheet (no spaces are allowed). Any descriptive text is for your reference only.
- Step 2:** For the Ticksheet Type, select the kind of ticksheet you are building from the directory tree, such as the Best & Worst ticksheet in the Relevance folder. (Click a plus sign to expand the tree.) The kinds of ticksheets are described in Ticksheet Types above.

Figure 26: Ticksheet Dialog Box: General Tab

Assigning the Ticksheet to a Dataset

Datasets are subsets of the constellation's ticksheets listed in the left frame of Clarity, Momentum, and Relevance Web pages.

Step 1: Click New Set to create a new dataset. The Dataset dialog box (Figure 27) is displayed. Enter the dataset name and the label name (the name as it should appear on the ticksheet). Enter any help text that will be displayed to end users.

After data sets have been created, you can select them from the drop-down list. (See "Setting Up Datasets" on page 213 for more information.)

Step 2: In the General tab, enter the dataset name (the name used internally by the system) and the label name (the name that will appear in the left frame of the ticksheet under the Dataset heading as shown in Figure 28).

Step 3: Enter any help text that will be displayed to end users in the Help Text box.

General Tab

Step 4: The Tickshéets tab shows the ticksheets assigned to this dataset.

Note: Datasets are organized in the Shared Interface folder in the EpiCenter Manager directory tree. To rearrange the order in which the Datasets appear to end users, right-click a dataset in the Shared Interface folder, and select Up or Down from the pop-up menu.

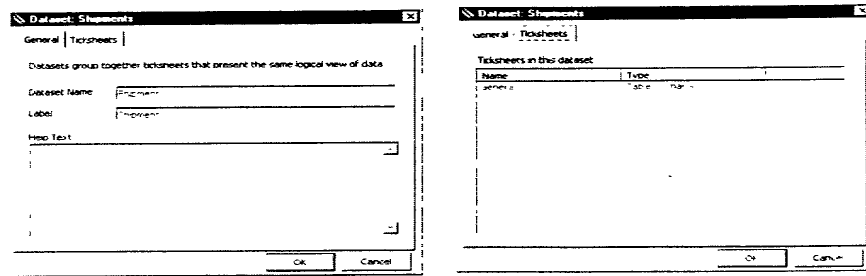
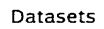


Figure 27: Dataset Dialog Box



Assigning Access to Groups and Users

New users and groups are set up using EpiCenter Manager's Security folder (see "Security" on page 200 for instructions). In the Access panel of the General tab of the Ticksheet dialog box (Figure 26), you will select which of those users and groups can view this ticksheet.

Attributes Tab

- To allow access to an individual user, click Add User and select the user from the Choose User dialog box.
- To allow access to a group, click Add Group and select the group from the Choose Groups dialog box.

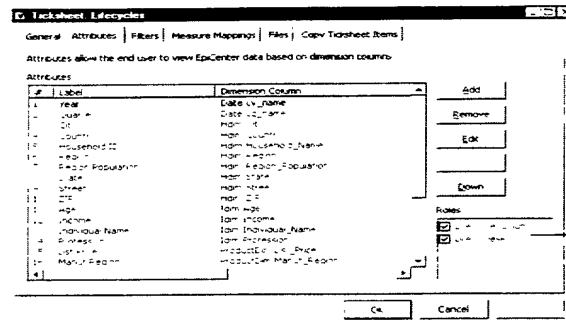
Note: Because the Access information is owned by users and groups, and not ticksheets, these assignments take effect immediately. Changes are not rolled back when you click Cancel in the Ticksheet dialog box.

Attributes Tab

In Clarity, the attributes, which derive from the dimension tables, are arranged by columns and rows. For example, if you select products for the column and years for the row, the report will have columns for each of the designated products in the database and rows for each of the designated years. In Relevance ticksheets, you may select attributes in order to forecast trends, or to analyze highs and lows, best and worst cases, or graphs for various aspects of your business.

You can use the Attributes tab (Figure 29) to set up the attributes for the ticksheet.

Attributes Tab



Assign attribute roles here

Figure 29: Ticksheet Dialog Box: Attributes Tab

- Step 1:** Click the Add button, which displays the Attribute dialog box (Figure 30).
- Step 2:** Enter the attribute's Label (the name that appears on the ticksheet).
- Step 3:** Enter an abbreviation that the system may apply if the Label name is too long (for example, ASP for Average Sales Price).

Attributes Tab

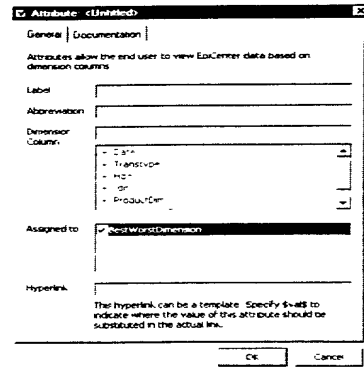


Figure 30: Attribute Dialog Box

- Step 4:** Select the dimension column for this attribute from the base dimension table listing. (Click a plus sign to expand the tree.)
- Step 5:** Assign attribute roles, such as use for both a column and row in a Clarity ticksheet, by selecting the role in the lower-right area of the dialog box (see Figure 29). (The ticksheet type determines these options: see “Ticksheet Types” on page 124.) You can also use the Roles panel of the Attributes tab to do this.
- Important:** Attributes can be defined only once per ticksheet. If you want a single attribute to appear in more than one place on a ticksheet, you must assign it multiple roles. (Clarity and some of the Relevance ticksheets have multiple roles.)
- Step 6:** The Hyperlink text box enables you to use the attribute as a link. For example, companies in a customer list could be linked to Web sites, such as <http://www.co-select.com>. You can use the string `val` to indicate where the attribute is to be substituted; for example: `www.val-select.com`.

Step 7: You can use the Documentation tab to enter a description for internal use only, or to make a glossary entry for this attribute.

Attributes and other ticksheet terms can be hyperlinked to a glossary page. When a user clicks a link, a glossary page displays which defines it. These glossary entries help users to understand your terminology. See “Setting Up a Glossary Entry” below.

Step 8: In the Attributes dialog box, click OK to add the attribute. It now appears in the list in the Attributes tab. Follow the same procedure to create (or modify) additional attributes.

Step 9: Select an attribute and click Up and Down in the Attributes tab to rearrange the order in which the attribute appears on the ticksheet.

Step 10: If you have not already done so, assign a selected attribute the application roles (such as ClarityColumn and ClarityRow) that are appropriate for the ticksheet type.

To modify an attribute, select it from the list in the Attributes tab and click Edit, which opens its Attribute dialog box.

09625518.072500

09625518.072500

Filters Tab

Setting Up a Glossary Entry

To set up a glossary entry for a ticksheet item, click New in the Glossary Entry area of the dialog box and type the entry name and help text (that displays to end users) in the Glossary Entry dialog box (Figure 31). Click OK.

Click Edit to open a Glossary Entry dialog box for editing.

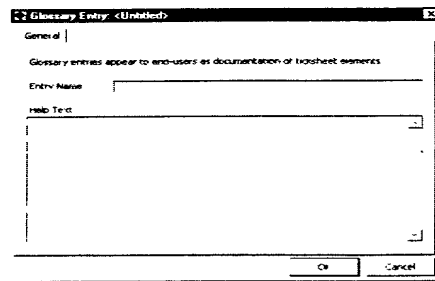


Figure 31: Glossary Entry Dialog Box

Filters Tab

Filters enable ticksheet users to restrict the data accessed for a query to specific attributes (values in dimension columns); for example, the data may be filtered so only the values for direct sales during the years 1990 through 1995 are returned.

You can use the Filters tab of the Ticksheet dialog box (Figure 32) to define the filters for the ticksheet.

After filters have been added, the information is displayed in the ticksheet's Filters window. You can double-click one of these listings to open its dialog box for editing, or select it and click Edit.

Filters Tab

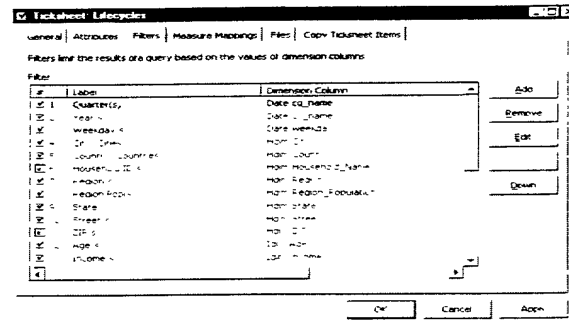


Figure 32: Ticksheet Dialog Box: Filters Tab

Follow these steps to define filters:

Step 1: Click Add to open the Filters dialog box (Figure 33), which has two tabs: General and Filter Elements.

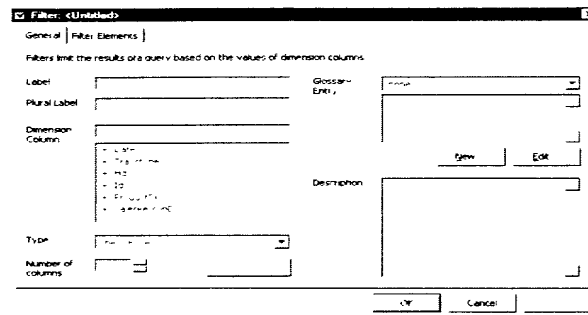


Figure 33: Filter Dialog Box: General Tab

006655318 072500

Filters Tab

- Step 2:** In the General tab, enter the name of the filter category as it will appear on the ticksheet, such as Fiscal Year.
- Step 3:** The plural value is the text that follows *All* as in All Fiscal Years on the ticksheet. The system enters this for you. Verify that it is correct.
- Step 4:** Enter any description for your own reference in the Description text box.
- Step 5:** To add a new glossary entry for this filter category, select New and enter a Glossary Entry as described in “Setting Up a Glossary Entry” on page 134, or select an existing entry.
- Step 6:** Select the dimension column for which data will be filtered from the dimension tables tree. Right-click to expand a plus sign. The selected dimension table name and column name are displayed in the Dimension Column textbox.
- Step 7:** Select the type of filter that the user will select in the Filters window of the ticksheet: a check box, a dynamic listbox, list members, or radio buttons, a (static) list box, or a text box. The contents of a dynamic listbox are updated when the Application Server starts. (Dynamic listboxes are refreshed; static listboxes never change.) List members are derived from Momentum ticksheets. (Momentum lists can be used as filter options in Clarity.)
- Step 8:** For the check box filter type, select the number of columns that the attributes will be divided into; for example, Fiscal Quarter has four columns. The filter user interface changes depending on the filter type.
- Step 9:** You can use the Filter Elements tab to create Filter Groups and Filter Elements (see Figure 34).
- A Filter Group* is a logical grouping of filter elements that applies only to check box filter types. For example, a filtering by year may have the Group Q197 (first quarter of 1997), with the months January '97, February '97, and March '97.
- A Filter Element* is a single check box for filtering within a Filter Group, or a single entry within a listbox.

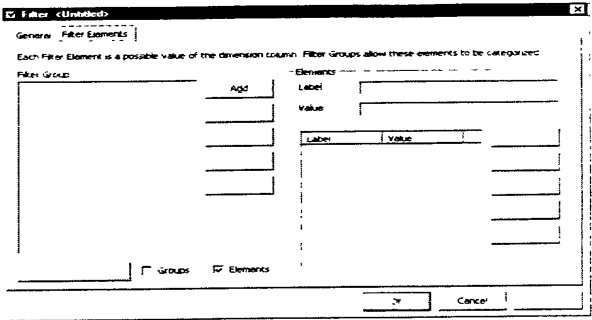


Figure 34: Filter Dialog Box: Filter Elements Tab

Defining Filter Groups

Filter Groups provide a convenience for ticksheet users. Creating a Filter Group does not modify the database. A sample check box Filter Grouping follows:

Group 1 — 1996: Q1 96, Q2 96, Q3 96, Q4 96 (*where Q1 96, Q2 96, and so forth are filter elements*)

Group 2 — 1997: Q1 97, Q2 97, Q3 97, Q4 97

Group 3 — 1998: Q1 98, Q2 98, Q3 98, Q4 98

You can use the Filter Group panel of the Filter Elements tab to define a Filter Group. Follow these steps:

- Step 1:** Click Add. The Filter Group dialog box is displayed (see Figure 35).
- Step 2:** Enter the label name (for the group name on the ticksheet) and any description for your reference.
- Step 3:** Click OK.

09625518.072500

09625548.072500

Filters Tab

Step 4: To add the filter group as a glossary entry, click New. Instructions are given in “Setting Up a Glossary Entry” on page 134.

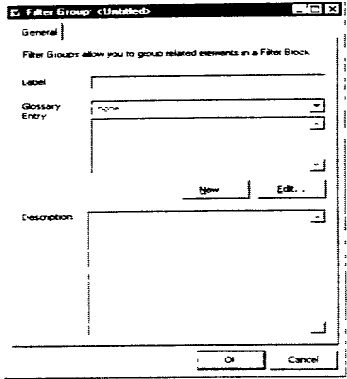


Figure 35: Filter Group Dialog Box

Defining Filter Elements

When entering filter elements, note that there is both a label and a value for each element. The value is the actual database value that will be filtered, whereas the label is what appears to the end user on the ticksheet *and* the report page. If the value *CA* has a label of *California*, then when the user clicks the *California* checkbox, the database query actually filters on *CA*. When the results are displayed, *CA* is replaced with *California*. This feature allows database values to be mapped both on the filter and display side.

To define Filter Elements for the group you created, follow these steps:

- Step 1:** Open the Filter Elements tab of the Filter dialog box.
- Step 2:** In the Elements panel, click Add. The Filter Elements dialog box (Figure 36) is displayed.

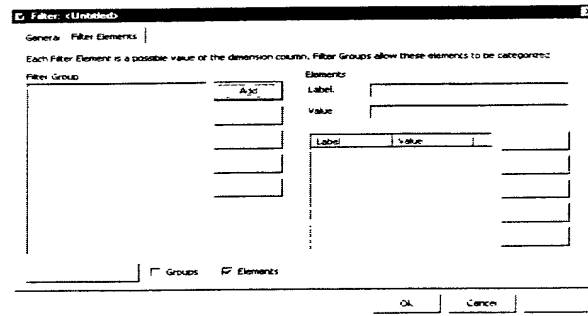


Figure 36: Filter Elements Dialog Box

Step 3: To have the system fill in the list using an SQL SELECT statement (to extract a field from a table), click Fill from Query. The SQL Query dialog box (Figure 37) is displayed.

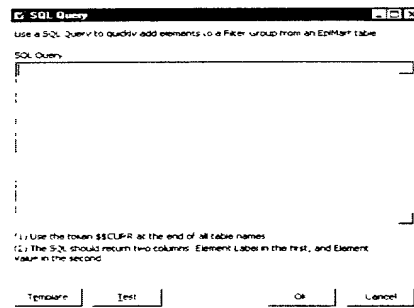


Figure 37: SQL Query Dialog Box

Measure Mappings Tab

- Step 4:** To have EpiCenter Manager create a template SQL to automatically fill your filtering query, click the Template button.
- Step 5:** Replace the column name and the table name for your data.
Append the token `_0$$CURR` at the end of all table names: for example, `Product_0$$CURR`.
- Step 6:** Click Test to see the query that this produces. This query has two columns by default: one for the element's label and another for the element's value.
- Step 7:** Click OK to begin the actual extraction.

Measure Mappings Tab

Note: Measure Mapping does not apply to Relevance Influence ticksheets. Relevance Influence uses measure sets, which are discussed in "Measure Sets" on page 156.

Measure mapping is a two-step process. First, you will add all of the measure column elements that will appear on the final ticksheet and arrange them by column. This sets up the organization of the Web page ticksheet and assigns names to the measure column elements in the ticksheet (the actual value is derived from the Measure to which you map this combination of elements).

Second, you need to map each set of measure column elements that a user might choose from each column in the ticksheet to the measure calculation that this choice represents.

The Measure Mapping tab also allows you to create a new Measure. Click New Measure, which opens the Measure dialog box (see Figure 25). Follow the instructions given in "Measures" on page 118.

Adding Elements to Columns

To add elements in the appropriate columns:

- Step 1:** Open the Measure Mappings tab (Figure 38) in the Ticksheet dialog box.
Note that a single empty column is displayed for a new ticksheet.
- Step 2:** Click Add to add the measure column elements for this column.

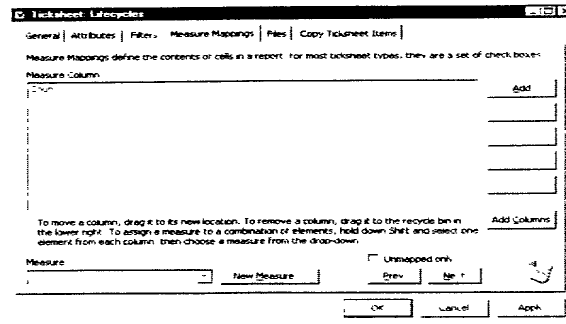


Figure 38: Ticksheet Dialog Box: Measure Mappings Tab

- Step 3:** You can use the Measure Column Element dialog box (Figure 39) to define the elements. Enter the label (the name as it will appear in the ticksheet column, an abbreviation (if appropriate), and a description for your reference.

Measure Mappings Tab

The screenshot shows a dialog box titled "Measure Column Element: (Untitled)". It has a "General" tab. Below the title bar, there is a note: "Each Measure Column Element is a checkbox on the ticksheet. A combination of an element from each column specifies a measure." The dialog contains several input fields: "Label" with the text "Units", "Abbreviation" with the text "U", "Glossary Entry" with the text "none", and "Description" which is empty. There is a "New" button next to the "Glossary Entry" field. At the bottom of the dialog are "OK" and "Cancel" buttons.

Figure 39: Measure Column Element Dialog Box

Step 4: To set up a glossary entry, click New and type the entry name and help text (that displays to end users) in the Glossary Entry dialog box (see Figure 31).

Measure column elements in ticksheets, such as Units, Gross, and Sell-Through, can be hyperlinked to a glossary page. When a user clicks a link, a glossary page displays which defines it. These glossary entries help users to understand your terminology.

Step 5: Click OK to add the measure column element to this column.

Step 6: Click the Add button again to add another measure column element to the column. Continue to add elements as described above until you have entered all of the elements for the column.

Step 7: Click Add Columns to add a second empty column. Click Add to enter the new column's measure column elements as described above.

- Step 8:** Click Add Columns to add a third empty column, if applicable. Add the measure column elements to the column. Add additional columns and measure column elements as appropriate.
- Step 9:** To change the order of one column with another (the order in which they display on the ticksheet), select a column and drag it onto the new column location. The columns' contents are switched.
- Step 10:** To remove a column, drag it to the recycle bin in the lower right of the dialog box.

Mapping Elements to a Measure

Users can select any combination of measure column elements (one per column) on a ticksheet. Each combination of elements equals a measure (whose calculations determine the contents of the generated report or query). For each ticksheet, you will need to map every measure column element in a given column to every other measure column element in all of the other columns (to cover all the possible combinations that a user may select).

To map elements to a measure:

- Step 1:** While holding down the Shift key, select an element from each column. Each selection is highlighted.
- Step 2:** With all of your selections highlighted, choose a measure from the drop-down list.

This measure will be invoked by the system whenever the user selects this combination of elements on the ticksheet.

Verifying that All Elements are Mapped

Click the Previous and Next buttons to cycle through the combinations of measure column elements that comprise a measure. The mapped element in each column is highlighted, and the measure that these elements map to is shown in the Measure text box.

09625548 072500

Reports Tab

Select Unmapped Only in the Measure Mappings tab to display only those combination of elements that have *not* been mapped. (Remember that each distinct path through the columns must be mapped.)

Click the Previous and Next buttons to cycle through the possible combinations. Unmapped elements are highlighted, and no measure name is displayed in the Measure text box.

Reports Tab

The Reports tab of the Ticksheet dialog box lists all reports saved for this ticksheet by report name, Report Gallery folder, date last modified, and the user who modified it.

Important: Whenever a new ticksheet is created, the administrator should save one report in the Default subfolder in the Report Gallery's Public folder (see "Report Gallery" on page 209) and make it the default. This allows check boxes to be set to reasonable values for all users when they first use the ticksheet.

You may select a report and click Remove to delete it. (If appropriate, you can delete the entire list.) You may want to delete "orphaned" reports, which are reports that cannot be accessed by anyone via Epiphany's front-end Application Server, and may occur when a user is deleted. Select the Only Show Orphans option in the dialog box to display orphaned queries for deletion.

Select a report in the list and click Edit to display the Report dialog box for this ticksheet for editing. (Figure 15).

COPY TICKSHEET ITEMS

You may use the Copy Ticksheet Items tab (Figure 40) of the Ticksheet dialog box to copy the attributes, filters, measure mappings, and measure sets (when working with an Influence ticksheet) from one ticksheet to another ticksheet within the constellation. *All* members of the selected items, such as all filters, are copied from the current ticksheet to the target ticksheet. You may, however, copy one or more of the ticksheet items; for example, copy only the attributes, or only the attributes and filters to the ticksheet and leave the measure mappings.

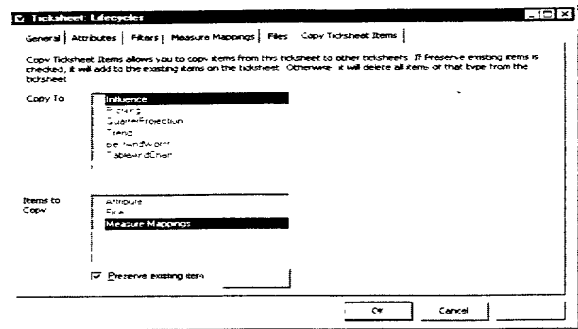


Figure 40: Ticksheet Dialog Box: Copy Ticksheet Items Tab

When *Preserve existing items* is selected in the tab, the copied items are added to the existing ones, creating duplicates on the target ticksheet. Thus if both the source and target ticksheets have a filter named *Channels*, after copying, the target ticksheet has two filters named *Channels*.

Important: If *Preserve existing items* is not selected, the copied items replace (delete) all members of the same item on the target ticksheet. Be sure that you intend to delete these.

Configuring Relevance Ticksheets

Preserving existing items does not work when you copy Measure Mappings. If, however, you are copying attributes, filters, and measure mappings, and *Preserve existing items* is selected, you are warned that the measure mappings will not be preserved. Then all of the other items are copied with preserve and the measure mappings are copied without preserve.

When the system copies measure sets (Influence ticksheets only), it attempts to link them to the same attributes as on the source ticksheet. If there is not one with the same name, it copies the set, but does not link it to any attribute.

To copy items from this ticksheet to another ticksheet:

- Step 1:** Select from the Copy To: box any ticksheet that you want to inherit items from this ticksheet.
- Step 2:** Select the items to be copied to these ticksheets.
- Step 3:** Click Go to begin the copy.

CONFIGURING RELEVANCE TICKSHEETS

The configuration of Relevance ticksheets, other than Influence ticksheets, (see “Relevance Influence Ticksheets” on page 152) is similar to that of a Clarity ticksheet. If you have created Clarity ticksheets that have similar attributes and filters to Relevance ticksheets, you can use the Copy Ticksheet Items tab of the Ticksheet dialog box (Figure 40, on page 145) to copy attributes and filters.

Each Relevance ticksheet has its own attribute roles. For example, the Best & Worst ticksheet has only one kind of role, a BestWorstDimension, and only attributes assigned this role appear in the ticksheet attributes area.

In general, there is no reason to conceal attributes on a Relevance ticksheet from users. The exception is when there are so many that the ticksheet becomes cluttered, or when an attribute has extremely high cardinality (many values). For queries involving such attributes, the Best & Worst or Influence query may be very slow.

Creating Default Relevance Ticksheets

When users first use Relevance ticksheets, they may be slightly confused by the many new kinds of analyses they can do. To make their experience with Relevance positive, follow these general rules when creating default Relevance ticksheets:

- The default ticksheet for a Relevance report should convey the value of the Relevance analysis; that is, it should be something that could not be accomplished via a Clarity ticksheet.
- The report should run in 10 seconds or fewer.

Guidelines follow for creating default Relevance ticksheets.

Best & Worst

The *Best & Worst* ticksheet has only the BestWorstDimension role only.

Choose at least three or four attributes. If you choose only two, the user may fail to see the value of Relevance over Clarity for this type of query. Best & Worst excels at analyzing multiple attributes.

Keep in mind when configuring a default Best & Worst ticksheet that end users can readily understand the comparison method “compare to another filter group.” They can look at reports that compare the most recent complete time period (for example, Q3 1998) with the previous complete time period (Q2 1998). In this case you would set the comparison filter to Q2 1998, and set the main filter to Q3 1998. For display options, you can select the top and bottom seven results, which shows enough data without being overwhelming. Choose the “one at a time” option to make the query run faster.

Try various attributes and evaluate the results in terms of an effective report.

Profiling

The *Profiling* ticksheet also has only one role, the ProfilingDimension. Only attributes assigned this role will appear on the ticksheet. Profiling is not affected by high-cardinality attributes as much as Best & Worst, so avoiding clutter is the main reason to not assign attributes to the ProfilingDimension role.

Creating Default Relevance Ticksheets

For the Profiles default report, you could select all attributes. If you do not select all attributes, at least attempt to produce a variety of chart types in the output. Note the following:

- Choosing any extremely high-cardinality attribute should generate an 80/20 chart.
- Choosing a high-cardinality attribute should generate a pareto chart.
- Low-cardinality attributes usually result in pie charts.
- A measure for which some attribute values have net negative totals can generate a stacked bar chart.

For the display options, select medium-sized charts.

Choose Auto for the units, and zero digits of precision.

Trends

The *Trends* ticksheet has two roles: TrendsRow and TrendsColumn. The TrendsColumn role should be assigned only to date attributes, such as Fiscal Quarter, Fiscal Year, and so forth. The Trends Row role may be assigned to any non-time attributes.

For a Trends default report, you can use the “straight line fit” with two periods forecasted.

Choose columns and filters such that about four or five time periods of actual data exist. (This way all of the projected columns will fit on users’ monitors.)

For display options, select yes for Include Charts. Auto for the currencies and quantities, and 0 decimal places. Do not select Include Sums. Set Percentages to neither. Select the top 10 rows, all columns, sort rows by amount, and set Filter Current Period to yes.

This last option means that if the last column of real data is an incomplete time period, for example, it is Q3 1998, and the last extract date is September 8, 1998, then Trends realizes that Q3 is incomplete, and will not use it in fitting a trend line to the data.

Quarter Projections

The *Quarter Projections* ticksheet has two roles: ProjectionRow and ProjectionColumn. The attributes for ProjectionRow should be time attributes only, such as Fiscal Quarter or Month. The attributes for ProjectionColumn should be relative time attributes only, such as Days until End of Fiscal Quarter, or Weeks until End of Fiscal Quarter, or Days until End of Month.

When creating a default report for Quarter Projections, follow the 10-second rule. This usually means building a special aggregate that includes Fiscal Quarter and Weeks til end of Fiscal Quarter, or possibly adding Weeks til end of Fiscal Quarter to an existing aggregate that includes Fiscal Quarter. You can use the same default options as in Trends.

Lifecycles

Lifecycles ticksheets are similar to Trends ticksheets. The LifecyclesColumn role should be assigned only to date attributes. The LifecyclesRow roles may be assigned to any non-time attribute.

Note: The columns in Trends and Lifecycle ticksheets should be absolute date columns, such as the fiscal year or the month of a particular year, such as January 1998. They should not be relative or cyclical columns such as month (that is, January, February, and so forth).

Creating an effective Lifecycles default report may take a little more work than the other Relevance ticksheets. Choose an attribute for the rows, such as Product. You may want to set filters so that there are not too many rows; for example, by filtering on Product Line so that the rows contain mostly different versions of the same product. Set the options similar to the way you would in Trends.

Create the report, and note the beginning dates for the rows. Any row whose beginning date is the same as the first time period included in the filter settings, or whose first time period is the first time period that data exists in the Epiphany system, is most likely already in the middle of its lifecycle, and should be filtered out. (Select all rows and then de-select those rows that were already in the middle of their lifecycles in the time period the report is analyzing.)

09625518.072500

Aggregates for Relevance

Aggregates for Relevance

For the most part, Relevance can use the same aggregates that would normally be built with Clarity. Some parts of Relevance, however, need Relevance-specific aggregates to prevent degraded performance. This section discusses the queries that are constructed when running Relevance, and the aggregates that should be built to make these queries run faster.

Quarter Projections

Typically, creating a Quarter Projection ticksheet involves constructing a new attribute that is not used elsewhere in the system, usually “Weeks til end of Quarter.” No aggregates exist on this attribute, and thus all Quarter Projection queries will access the base dimension table, and take an extremely long time to run. Build an aggregate on the QuarterProjectionColumn and QuarterProjectionRow dimensions, in addition to a few other dimensions that might be commonly used as filters for Quarter Projections reports.

Profiling

Profiling queries involve a single attribute at a time. Ideally, each attribute used in the profiling ticksheet has its own aggregate, or it is included in other small aggregate tables.

Often it can be valuable to include high-cardinality attributes such as Customer or Product on a profiling ticksheet. For Clarity, you probably would not build aggregates involving these attributes. Clarity would never use one of these attributes by itself, but rather in combination with one of many other attributes, and the resulting aggregates might be too large. Because Profiling looks at individual attributes, however, it can make sense to build an aggregate even on a high-cardinality attribute.

Influence

Since Influence is used to discover the factors that most affect particular aspects of a business, aggregates should be built based on these business entities (that is, the primary dimension, such as customers or products).

First, there needs to be a fact table constructed that contains one row per business entity (denoted by the inclusion of a key to this entity). Each such row should also contain both the attributes that appear on the Influence ticksheet, as well as various measure sets (that is target variables). For example, there may be one row per customer in such an aggregate, and the columns could denote various demographic attributes, as well as various target variables, such as whether or not the customer bought particular products. Note that this may be a large table.

Influence will also make use of aggregates built on the table described above. Such aggregates should include columns such as the primary dimension key, some subset of the ticksheet attributes that seem most likely to affect the business and the measure sets of interest. For more information about Relevance, see “Relevance Influence Ticksheets” on page 152.

Best & Worst

Best & Worst has two methods for analyzing data. The first method looks at attributes individually. In this case, Best & Worst can greatly benefit from having single attribute aggregates (similar to the case of Profiling). The second method looks at combinations of attributes. Here it is useful to have aggregates that contain subsets of related attributes. For example, Education and Income may be highly correlated demographic attributes and should probably be included in the same aggregate.

Trends and Lifecycles

Both of these applications use aggregates that you have probably already built for Clarity, since they essentially run a restricted kind of Clarity report (where the column is always a date dimension and the rows are always non-date dimensions) and then do some post-processing.

RELEVANCE INFLUENCE TICKSHEETS

Creating an Influence ticksheet is similar to creating Clarity and other Relevance ticksheets. All of the ticksheet tabs are set up in the same way. Influence has a specific Measure Sets tab, and no Measure Mapping tab.

How Influence Works

To answer end-user queries, the Influence application builds models that use two types of trees: classification trees and regression trees. A *classification tree* finds rules that can predict the value of a particular discrete-valued *attribute* based on the values of a set of other attributes, and a *regression tree* finds rules that can predict the value of a particular numeric *measure* based on the values of a set of attributes.

Classification Trees

You can use Influence to answer questions such as “Given the attributes of an individual, predict whether that individual will respond to a marketing campaign.” or “Given the attributes of a customer, find the customer segment that customer belongs to.” Influence answers these kinds of questions by building a model—a classification tree—that tries to predict the value of a particular attribute based on the values of a set of other attributes. This is termed *classification* because Influence attempts to predict which class someone or something will belong to; for example, it attempts to classify individuals as responders or non-responders, or as Tier 1, Tier 2, or Tier 3 customers.

The models built by Influence can also provide the answer to related questions about the importance of certain attributes. An Influence classification tree that classifies individuals based on how likely they are to respond to a marketing campaign also provides answers to the question, “Which attributes of an individual have the most influence on whether that individual will respond to a marketing campaign?”

Regression Trees

You can also use Influence to answer questions such as “Given the attributes of a customer, predict the total amount of revenue that customer will generate.” In this case, the prediction is a measurement relating to the customer, not an attribute. The model that Influence builds to answer this type of question is known as a *regression tree* because it performs regression in the statistical sense; that is, attempts to predict the value of one variable (total revenue) given the values of others (the attributes of the customer).

Within Epiphany, the difference between classification and regression is that classification trees use attributes to predict an attribute, and regression trees use attributes to predict a measure. The basic questions answered by the two types of trees are very similar (as are the techniques used by Influence to build them), but they require slightly different configuration.

Setting Up an Influence Ticksheet

Before creating an Influence ticksheet, determine:

- Step 1:** Whose behavior you want to predict: this is the *primary dimension*.
- Step 2:** What you are trying to predict about the primary dimension: this is the *target*.
- Step 3:** The attributes needed to make the prediction; these are the *source attributes*.

For example: “How do the *age*, *income*, and *city of residence* of a customer influence whether that customer will *buy an extended warranty*?” The primary dimension is Customer, the target is the attribute Bought Extended Warranty, and the source attributes are Age, Income, and City of residence.

09625518.072500

Setting Up an Influence Ticksheet

Determining the Primary Dimension

The *primary dimension* is the dimension role that represents the entity about which Influence will make predictions.

The first step in configuring an Influence ticksheet is to determine the *single* primary dimension. (Each Influence ticksheet must have one and only one primary dimension, although it may have multiple targets and multiple source attributes.)

Note: When Momentum is also installed, the primary dimension for Influence is often identical to the Momentum individual or household dimension.

You do not have to explicitly specify the primary dimension in EpiCenter Manager. The choice of the primary dimension, however, restricts the allowable targets and source attributes.

Determining the Target

The target of an Influence query is the *value* that the Influence model attempts to predict. You will define targets in EpiCenter Manager using the Measure Sets tab (see Figure 42) of the Influence ticksheet dialog box.

A *measure set* is a named collection of measures. Measure sets are used to define the target variables you care about predicting when using Influence. A measure set includes the target attribute you would like to predict, as well as a measurement on that attribute. All measure sets include a Count measure, and most measure sets additionally include either a Sum measure or an attribute. See “Measure Sets” on page 156.

Determine the Source Attributes

Source attributes are the attributes whose values Influence uses to predict the value of the target for each member of the primary dimension.

For any row in the primary dimension (for example, an individual), there must be a unique value of any source attribute. Obviously, for attributes from the same dimension, there will be only one value. For attributes from another dimension, care must be taken to ensure that this relationship holds. For example, if the primary dimension is individual, one would not choose Product as a source attribute because individuals can buy many products, so it would not be the case that for any individual there would be a single value of the product attribute.

When the primary dimension “rolls up” to another dimension role, source attributes from that dimension role are also acceptable. For example, suppose there is an Individual dimension role and a Household dimension role. If the primary dimension is Individual and no Individual belongs to more than one Household, then Individuals “roll up” to Households and attributes from the Household dimension role can be used as source attributes. In general, source attributes that are not selected from the primary dimension must not have a one-to-one or one-to-many relationship to the primary dimension.

Important: When configuring an Influence ticksheet that has source attributes that are not from the primary dimension, ensure that, because of the properties of your data, the appropriate relationship between source attributes and the primary dimension holds. EpiCenter Manager does not enforce the requirement of a one-to-one or one-to-many relationship, but if this type of relationship does not exist, then some or all Influence queries may fail.

You will use the Attributes tab of the Influence ticksheet dialog box to select the source attributes from a list of the existing attributes. (If appropriate attributes do not exist, you need to create them as you would any other attribute.) Each source attribute must be assigned the TreeDimension attribute role. Attributes assigned to this role are available to the user for inclusion in the set of attributes whose values are used as input data for classification or regressions.

In some cases, you may also want to include attributes on your Influence ticksheets that are not assigned to any attribute roles. For example, to give users the option to build a classification tree to predict the value of an attribute that is not allowed to be used as one of the set of predictor attributes. These attributes need to be on the Influence ticksheet, but should not be assigned to the TreeDimension attribute role.

Setting Up an Influence Ticksheet

Measure Sets

A measure set consists of selected measures that have an assigned name. Each measure within a measure set is also assigned the measure role of Count or Sum. The meaning of a measure role depends on whether the measure set is used for classification or regression.

To define a measure set for classification (a classification target):

- Step 1:** Open the Measure Sets tab of the Influence ticksheet and click Add.
- Step 2:** Click the New button in the Choose Measure dialog box. The Measure Sets dialog box is displayed (Figure 41).
- Step 3:** Enter the name of the measure set.

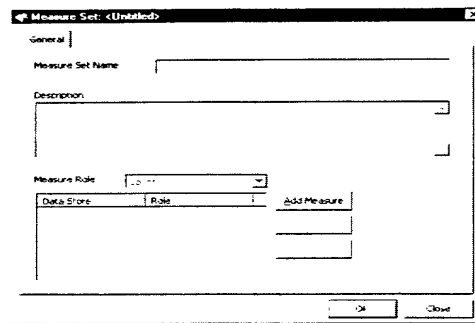


Figure 41: Measure Set Dialog Box

- Step 4:** When defining a measure set for a list membership target (that is, a measure set with no attribute and only the Count role defined), you can enter a short description in the Description text box that will show up in the Influence ticksheet target selection box. This is useful for cases in which the list membership target is weighted by a measure that is not the count. For example, if the user associates Revenue with the Count role,

then he or she may want to enter “weighted by revenue” in the description box of the measure set. Then, the target will show up in the Influence ticksheet as Member of List, weighted by revenue.

Step 5: Select Count from the Measure Role drop-down list. (The Sum measure role is not used for classification.)

Step 6: Click Add Measure and choose a measure from the list that gives a weighted count of the members of the primary dimension. (This measure must have already been created using the Measure dialog box as described in “Defining a Measure” on page 118.)

For example, choose a measure that computes a simple count of the number of units shipped, or a measure that sums the price of all products shipped, which produces a count that is weighted by price.

Step 7: Click OK.

Step 8: You must associate an attribute with this target (see “Setting Up Targets for the Ticksheet” on page 158).

Defining a measure set for regression (a regression target) is similar to defining a classification target. When you select a measure role for a measure set:

- Select a measure for the Count measure role that counts the number of members of the primary dimension.
- Select a measure for the Sum measure role that is the sum of the fact column that corresponds to the fact value.

The Measure Sets tab (Figure 42) allows you to specify targets for classification or regression by choosing measure sets and, optionally, associating them with attributes. Influence decides whether a measure set is for classification or regression based on whether there is an attribute associated with the measure set or not.

Setting Up an Influence Ticksheet

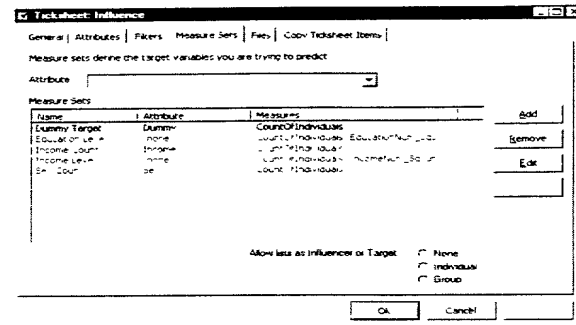


Figure 42: Influence Ticksheet Dialog Box: Measure Sets Tab

Setting Up Targets for the Ticksheet

After defining measure sets, you need to set up classification and regression targets for the ticksheet:

Step 1: In the Measure Sets tab, select a measure set from the list.

Step 2: *For classification targets only:* select an attribute to be associated with this measure set from the Attribute drop-down list.

The attribute whose value is being predicted *must* belong to the primary dimension. The measure you select for the classification target's measure set should give a weighted count of the rows in the fact table.

Note: Regression targets are not associated with an attribute because the value being predicted is a measure value rather than an attribute value.

Step 3: Clicking Update sets this attribute as the target for the selected measure set and makes it available to the ticksheet.

Step 4: Configure the other Influence ticksheet tabs following the instructions given for setting up Clarity and Relevance ticksheets.

Tips for Using Influence

Some tips related to using Momentum lists, slowly changing dimension, and aggregates with the Influence application follow. There is also a suggestion for avoiding longer query time.

Using Momentum Lists as Influence Attributes

When the primary dimension of an Influence ticksheet is the same as the Momentum individual or household dimension, Influence can be configured to allow the user to create lists of individuals or households in Momentum and then treat membership or non-membership in a Momentum list as a binary attribute for the purpose of Influence queries.

To specify whether individual lists, household lists, or no lists at all are allowed on a particular Influence ticksheet, use the *Allow lists as Influencer or Target* option on the Measure Sets tab (Figure 42).

For example, you could create a list of all the individuals who bought Product X in the last fiscal year and use that list of individuals as a source attribute in Influence to find out whether buying Product X is a good predictor of some other attribute of the individual (such as buying a related Product Y).

Membership in a Momentum list can be used as either a source attribute or a target attribute for a classification tree. Allowing individual lists to be used as attributes is a legitimate choice *only* when the primary dimension of the Influence ticksheet is individuals, and allowing household lists is a legitimate choice *only* when the primary dimension of the Influence ticksheet is households. You can always disallow lists from being used as attributes.

To add the option of using list membership as a classification target to a ticksheet that has Individual or Group selected for the *Allow lists as Influencer or Target* option in the Measure Sets tab, create a measure set that has only the Count measure role defined (not Sum), as with a regular classification measure set. Do not associate the measure set with an attribute. Measure sets that do not have the Sum measure role defined and are not associated with an attribute are implicitly associated with membership or non-membership in a Momentum list specified by the end user of an Influence ticksheet.

Using Momentum

Influence and Slowly Changing Dimensions

When the primary dimension of an Influence Ticksheet uses the Slowly Changing Dimension semantic type, a single logical member of the dimension (such as a single individual) may appear as multiple rows in the dimension table if the attribute values of that dimension member have changed over time (for example, an individual got married or moved to a different state).

In this case, Influence treats each row in the dimension table as a different member of the dimension when building its model. This may occasionally lead to minor differences in counts between Influence and Momentum, but should not present any problems.

Influence and Aggregates

If you build an aggregate on the primary dimension of an Influence ticksheet that includes all columns of the base dimension (or at least all columns that are used as attributes or filters on the ticksheet), Influence queries almost always utilize it.

Performance Issue

Two kinds of queries cause Influence to adopt a more complicated query plan: queries involving membership in Momentum lists as source attributes or targets, and regression queries that are filtered on dimensions other than the primary dimension. All other things being equal, these kinds of queries tend to take somewhat longer than ordinary Influence queries. If the processing time for Influence queries is a concern to your installation, you may want to avoid these two types of queries.

USING MOMENTUM

Momentum allows end users to generate lists drawn from demographic data for groups and individuals, such as Customers, Resellers, Sites, Households, Individuals in Households, Contacts, or other similar entities. (Note that the terms *groups* and *households* are synonymous in Momentum.)

Before you can configure a Momentum ticksheet, you need to set up one base dimension table that will contain the group demographic information for Momentum. Configure this base dimension table in the same way that you would any other base dimension table (see “Base Dimension Tables” on page 98).

As an option, if you want to configure ticksheets that return information related to individuals, in addition to group demographics, define a demographic base dimension table for this individual data. For example, in a fund-raising campaign, you may have data related to households that contribute, in addition to data for the individual contributor. In this case, you would set up two base dimension tables: one for group and one for individuals.

Each EpiCenter may have one Momentum constellation. The Momentum constellation has a built-in dimension role called *group* that points to a single base dimension table that holds group demographic data. The dimension role name for individual demographic data is *indiv*. (The dimension role names cannot be modified.) The *group* and *indiv* dimension roles must point to separate base dimension tables.

After setting up your group base dimension table, and optionally, an individual base dimension table, you are ready to open the Momentum constellation. Mini-dimensions, which are analogous to aggregates, need to be set up in the Column Sets tab of the Base Dimension dialog box. You can set these up later.

Follow these steps the first time you work with the Momentum constellation:

Step 1: Right-click the Constellations folder in the EpiCenter directory tree and select New Constellation from the pop-up menu.

Note: The Momentum Constellation folder will appear in the EpiCenter Manager directory tree only if the Momentum application has been installed, and *Include Momentum* was selected in the Initialize EpiCenter dialog box.

Step 2: In the New Constellations dialog box, select Momentum and click OK.

Using Momentum

- Step 3:** Select the base dimension table that you intend to use for Momentum group demographic data from the list of your base dimension tables. The Momentum constellations folder that is displayed has sub-folders named Facts, Dimension Roles, and Ticksheets.
- Step 4:** Open the Facts folder, which has one fact called *Ind_Group_Joiner*. This is a special fact table for Momentum that serves to connect the individual and group base dimension tables. It does not function in the same way as other fact tables in the Epiphany system.
- Although you cannot modify this dialog box, you do need to define how to extract data into the *Ind_Group_Joiner* table. For more information, see “The Ind_Group_Joiner Fact Table” on page 164.
- Step 5:** Open the *group* Dimension Role dialog box. Select the Momentum tab and assign a label and a plural name for this demographic dimension. These are the names that appear on the Momentum ticksheet for end users.
- Step 6:** If you plan to work with individual demographic data and have set up a corresponding base dimension table, right-click the Dimension Role folder and select New from the pop-up menu. A Dimension Role dialog box for the role *indiv* is displayed. Select this base dimension table from the drop-down list.
- Step 7:** Select the Momentum tab and enter the label and plural names for the individual entity that will appear to end users.
- Step 8:** The Options Labels tab of the Configuration dialog box has option names and labels that configure special words that display to end users on Momentum ticksheets. See “Option Labels” on page 290 for instructions on customizing these items.

Step 9: Momentum provides a quick count feature in which you can query a subset of the data, which makes the queries run faster by that factor. You can turn on this feature by assigning a positive value to *min_sample_invlog10*, which is a configuration option in the General tab of the Configuration dialog box (see “*General Settings*” on page 285). The *min_sample_invlog10* option is the inverse log (base 10) of the sampling probability. For example, a value of 2 would create a 1 percent sample ($1/10^2$).

Notes:

The Ticksheet Types tab of the Configuration dialog box (Figure 82, on page 294) enables you to disable an entire application by choosing the application from the drop-down list and de-selecting Enabled. This is useful if you want to test Clarity and Relevance ticksheets, but have not set up Momentum completely.

As mentioned, the columns you define in metadata are moved into the Momentum tables only if you have selected the option *Can be used in Momentum filter* for dimension columns, and the option *Queryable* for fact columns. Any changes to these options show up after you have run MomentumBuilder, which builds the Momentum tables. (You can run the Scrutiny debugging tool to verify if the Momentum tables are consistent with the metadata; see “Running the Scrutiny Debugging Tool” on page 223).

The Ind_Group_Joiner Fact Table

The Ind_Group_Joiner Fact Table

The *Ind_Group_Joiner* fact table defines the many-to-one relationship between individuals and groups. This table identifies the Momentum demographic dimensions, assigns individuals to groups, and defines the individuals and groups in Momentum. Each of these is discussed below.

- Identifying the Momentum demographic dimensions
MomentumBuilder and the Application Server use the *Ind_Group_Joiner* fact table's special dimension roles *indiv* and *group* to identify the individual and group base dimension tables.
- Assigning of individuals to groups
The data in this fact table determines which individual belongs to which group.
- Defining who exists for Momentum
The table also defines the individuals and groups that are a part of Momentum. Any individual or group that does not have its key in this table does not exist as far as Momentum is concerned. Even if you have individuals that belong to no group or to groups without individuals, you still need to enter their keys in this table for Momentum to recognize them.

How to Populate the Ind_Group_Joiner Table

When populating this table, consider these three cases:

1. Individuals who belong to a group and groups that have individuals. For each individual, assign an entry for the individual and the group that it belongs to. Assign an individual to one group only. This is the most general case and almost all of your individuals and groups will fall into this category.
2. Individuals who do not belong to a group according to the available data. Assign all individuals to the UNKNOWN group.
3. Groups with no individuals. Place an entry for each of the group and for the individual key assign it as UNKNOWN. Now the individual UNKNOWN is a part of multiple groups, but MomentumBuilder is aware of this situation, and it will handle these rows correctly.

Note: In the case of a slowly changing dimension, only the latest attribute is used, unless the attribute comes from a leaf dimension, in which case the attribute at the time of the transaction is used. *Leaf dimensions* are part of adjacent constellations (see “Why Use Adjacent Constellations?” on page 166).

The Momentum Constellation

The constellation you just set up is similar to the one shown in the diagram in Figure 43.

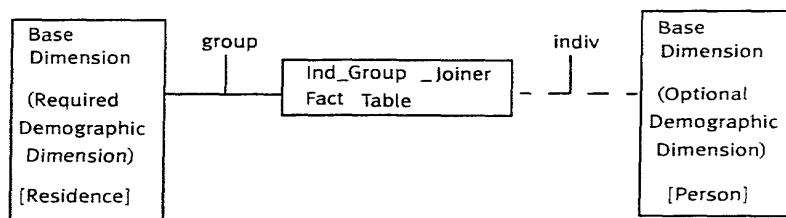


Figure 43: The Momentum Constellation

A user of a Momentum ticksheet applies filters to restrict demographic data; for example, to restrict the list further, or to include others in the list. After the user selects a filter in Momentum, he or she can find out how many matches it corresponds to in the database, either exactly or approximately.

The Momentum constellation shown above allows you to configure a ticksheet that can filter group and individual attribute data. The filtering occurs only on the demographic base dimension tables. Optionally, a Momentum ticksheet can be configured to allow users to filter behaviors, or transactions related to demographic attributes upon which the user has filtered. A *transaction* is usually an action such as bought, returned, called call center, received promotion or responded to campaign.

The Momentum Constellation

To configure a ticksheet that allows users to filter on transactions (that is non-Momentum fact table, such as Order), in addition to attribute data, requires that you set up additional constellations, called Momentum Adjacent constellations. These are optional, but their presence gives you access to Momentum's most powerful list making capabilities.

Why Use Adjacent Constellations?

The Momentum constellation for an EpiCenter is usually associated with other constellations in the EpiCenter, which are called *Momentum Adjacent constellations* (see Figure 44). While Momentum constellations are used for demographic queries, adjacent constellations may be used for behavioral and transactional queries. The information that can be queried is more complex because filtering can occur using Fact tables and non-demographic base dimension tables.

You may set up any number of adjacent constellations in your EpiCenter. Momentum Adjacent constellations can be queried by Clarity similar to any other constellation.

For example, a Momentum query based on a Momentum constellation is capable of returning a list of household members who satisfy a demographic criterion, such as sharing the same Zip code. When an adjacent constellation is also queried, the members of these household who responded to a request for a donation in a certain year could be determined.

Setting Up Momentum Adjacent Constellations

A Momentum Adjacent constellation must have at least one dimension role that points to one of the demographic dimensions (in Figure 44, the *Person role* dimension points to the Person base dimension table). Non-adjacent constellations are constellations that do not have an associated dimension role, and thus are not used by Momentum.

Additional Momentum Configuration

The dimension tables in a Momentum Adjacent constellation that point to non-demographic base dimensions are called *leaf dimensions*. These can be queried by Momentum and filtered via their presence in transactions using standard Clarity-style filtering. In Figure 44, the leaf dimensions are Sales Regions, Sales People, and Products.

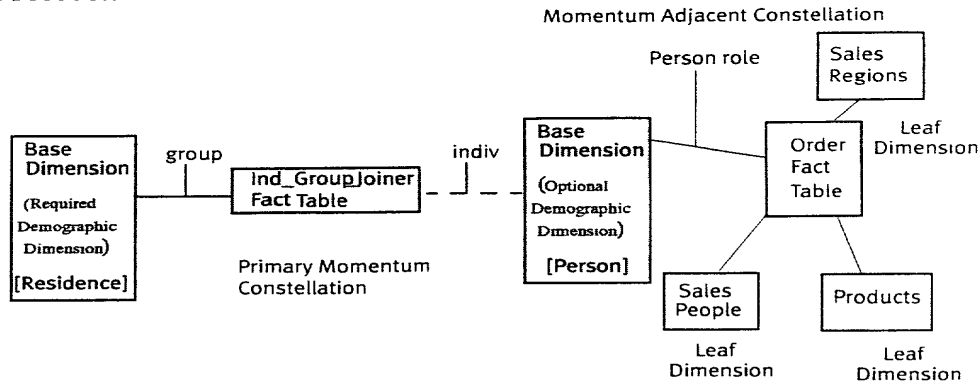


Figure 44: Adjacent Constellation Associated with a Primary Constellation

To set up an adjacent constellation, first set up the Momentum constellation. Then set up a constellation that has the fact table and dimensional attributes that you want to query for demographic data. At least one dimension role must connect the adjacent constellation's fact table to one of the Momentum demographic base dimension tables. In the Figure 43, this dimension role is the *Person role*.

Additional Momentum Configuration

Setting up a Momentum constellation also requires you to—

- define mini-dimensions
- build clusters and counts
- define transaction filters

Mini-Dimensions

Attributes of a dimension can be divided into disjoint sets such that attributes within a set are highly correlated, and there is very little correlation between an attribute from Set A and an attribute from Set B. These different sets of attributes are placed into different tables called mini-dimensions. The mini-dimensions are very small because the attributes are highly correlated, and since the correlation between different mini-dimensions is very low, their join gives a large dimension. Mini-dimensions compress the storage for a table and consequently reduce the time required to bring it up in memory when scanning the table for querying. The storage compression is significant for large demographic dimensions.

Clarity and Relevance have aggregate tables that speed up queries. You define these on the Column Sets tab of the Base Dimension dialog box where you also set up mini-dimensions. The fact that these are discrete allows you to logically separate sets used for Clarity and Relevance from those used by Momentum. There are also other differences between aggregate column sets and mini-dimensions:

- The mini-dimensions for a single base dimension table must not overlap. They are disjoint, whereas column sets may overlap.
- The resulting mini-dimension tables are mapped to integers, which results in mini-dimensions that are thinner than aggregates.

But, most importantly, you design aggregate column sets not only because attributes are correlated in the database, but also because end-user queries need all of these columns. Any single Clarity query can use only one column set. For mini-dimensions you still want data-correlated attributes in the same mini-dimensions, but because a Momentum query can use more than one set, the end-user correlation is not as important.

For example, assume that a customer base dimension table has columns for City, State, Zip, Occupation, Income, and Age. Data correlated attributes are—

Step 1: City, State, and Zip

Step 2: Occupation and Income

Step 3: Age

You can set up mini-dimensions on each of these three divisions. Since users tend to query number 1 and 2 at the same time, you could set up an aggregate on these.

You need to define mini-dimensions both on your demographic base dimension tables, and on the leaf base dimension tables. Mini-dimensions need to be defined in order to set up fact clusters and counts.

Guidelines for setting up mini-dimensions follow:

- Place only queryable attributes in mini-dimensions.
- Group attributes that are highly correlated in real life, such as zip code and area code.
- Place most queryable attributes in a mini-dimension; however, do not place an attribute in a mini-dimension that has high cardinality (many values), such as a name.
- An attribute can appear in one mini-dimension only.
- Because a mini-dimension can hold a maximum of 32,767 rows, create mini-dimensions that are smaller than this size.

Transaction Filters

Assume that a demographic entity, such as People, exhibits a behavior or transaction, such as Bought Product that you want to find out more about. For example, What product did they buy? Whom did they buy it from? When did they buy it? All of these questions can be answered by filtering on the leaf dimensions associated with the Order fact table using regular Clarity-type filters, which you define in the same way as a Clarity filter. The *what*, *who*, *when* types of queries that end-users might have are derived from dimension attributes.

You might also want to know *how much* these people bought, which is a numeric value, or a measure. To enable end users to apply a filter on measure data in this manner, you need to define transaction filters in addition to regular filters.

Transaction filters are always associated with a set of regular filters and, optionally, can be configured to apply a measure filter to the result of that filter set.

Clusters and Counts

Defining a transaction filter involves selecting—

- a fact table.
- the appropriate dimension role (the system is flexible and there may be more than one demographic dimension role pointer).
- the transaction type (fact tables can have more than one).

Clusters and Counts

Clusters and counts make transaction filtering faster. A *cluster* is a copy of the fact table sorted on an attribute. A large table should be sorted on disk with its leading term being the most selective filter.

Counts keep statistics about how often the mini-dimension row appears in the fact table. Counts are used by the Momentum query engine to select the best clustered copy of the fact table. (A cluster is useless without a count.)

When you associate a fact table in a Momentum Adjacent Constellation with a mini-dimension, the system creates copies of the fact table clustered on a certain leaf mini-dimension key. Clustering around a mini-dimension speeds up query performance because the fact table rows that need to be accessed are physically contiguous, thereby limiting the number of disk blocks that need to be scanned.

Although clustered copies are very useful, keep in mind that these are copies of the fact tables, and too many of these copies would consume disk space and increase the time required by MomentumBuilder to build the Momentum tables. In contract, counts are cheap.

For example, users typically ask about buying patterns by customers (demographic dimension) and product lines (attribute leaf dimensions). Define a mini-dimension on Product Line and related attributes, and cluster and count the Order fact table by that mini-dimension.

Guidelines for clusters:

- To be able to cluster a fact on a dimension, you need to specify at least one mini-dimension. You can assign all the queryable attributes for that dimension to a single mini-dimension.

- You need to specify at least one cluster on a fact table for it to be included in Momentum.

CONFIGURING A MOMENTUM TICKSHEET

Similar to Clarity and Relevance ticksheets, you will use the General tab of a Momentum Ticksheet dialog box to name the ticksheet, select the ticksheet type, and assign the ticksheet to a dataset. The ticksheet type is either *Filter Households* or *Filter individual*, and the dataset is always called Momentum (this is built-in).

Setting Up Momentum Attributes

The Momentum attributes are derived from the group or individual demographic base dimension tables.

You will set up attributes for Momentum ticksheets using the Attributes tab of the Momentum Ticksheet dialog box. The only difference between a Momentum attribute and Clarity or Relevance attributes is the kind of attributes allowed. If the ticksheet type is *Filter Household*, then only group attributes are allowed. If the ticksheet type is *Filter Individual*, the attributes may relate to both the group and individuals in that group.

The Role types are *FilterIndividualCols* for *Filter Individual* types and *FilterGroupCols* for *Filter Household* types. These are the columns available for inclusion in the actual lists.

00625548.072500

Configuring a Momentum Ticksheet

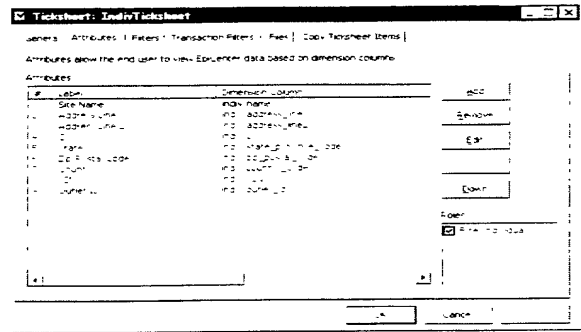


Figure 45: Momentum Ticksheet Dialog Box: Attributes Tab

Filters

All filters for the Momentum ticksheet—demographic filters and those used in conjunction with transaction filters—must be defined in the Filters tab. After filters are defined, they are “stored” here. (Think of this tab as a kind of warehouse of filters.)

Important: In order for a dimension column to be used as a Momentum filter, the *Can be used in a Momentum filter* option must be selected in the Dimension Column dialog box (Figure 17, on page 101).

For demographic filters, if the ticksheet type is *Filter Household*, then only group attributes are allowed: that is dimension columns from the *group* base dimension table. If the ticksheet type is *Filter individual*, then attributes may relate to both the group and individuals.

Transaction filters, which apply only to Momentum Adjacent constellations, are discussed below.

Setting Up Clusters and Counts on Fact Tables

Clusters and counts apply to Momentum Adjacent constellations only. You can use the Momentum tab in the Fact Table dialog box (Figure 46) to set these up. (Mini-dimensions need to have already been created for the leaf dimension tables.) Each selected mini-dimension is a cluster.

To set up a cluster:

Step 1: In the Clusters panel, click Add Dim.

Step 2: Select a mini-dimension from the Choose dialog box, and click OK.

This mini-dimension is added to the cluster list. (The numbering is for internal use only.)

Step 3: Continue to add additional mini-dimensions.

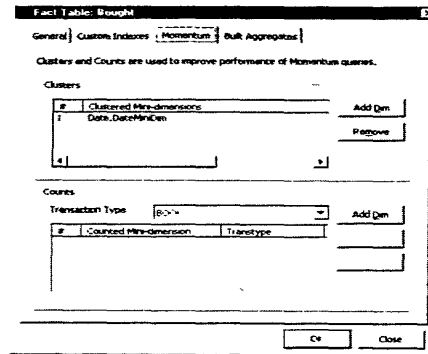


Figure 46: Fact Table Dialog Box: Momentum Tab

09625510.072500

Configuring a Momentum Ticksheet

As a rule, if you are building a cluster, also build a count for that mini-dimension. Build Counts on each transaction type that will be queried in the fact table.

To build a count for this cluster:

- Step 1:** Select a transaction type from the pull-down menu to associate with the mini-dimension.
- Step 2:** Click Add Dim.
- Step 3:** Select the mini-dimension to be counted from the Choose dialog box, and click OK.

Defining Transaction Filters

You can use the Transaction Filters tab (Figure 48) to define transaction filters.

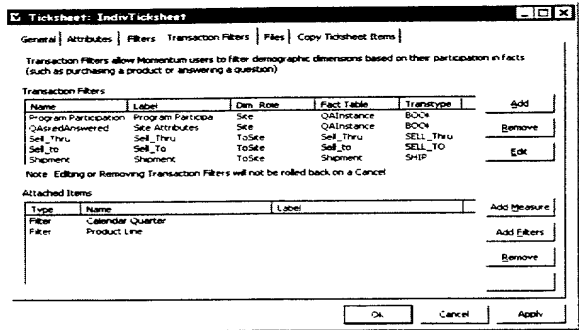


Figure 47: Momentum Dialog Box: Transaction Filters

To define a transaction filter:

- Step 1:** In the Transaction Filters panel, click Add, and click New (or select an existing filter).
- Step 2:** For a new transaction filter, using the Transaction Filter dialog box, enter the name and the label as it should appear on the ticksheet. You may also enter a description for your reference.

- Step 3:** Select the name of the adjacent constellation.
- Step 4:** Select a dimension role that connects the adjacent constellation to the demographic base dimension table.
- Step 5:** If there are multiple fact tables in the adjacent constellation, select the fact table associated with the dimension role.
- Step 6:** Select the transaction type. A fact table may contain multiple transaction types, but a transaction filter applies to only one.

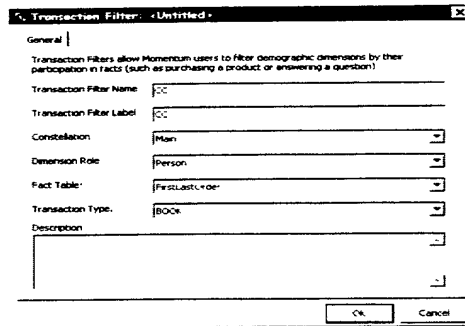


Figure 48: Transaction Filter Dialog Box

To associate a measure in the constellation with a transaction filter:

- Step 1:** Select the filter in the upper panel of the Transaction Filters tab.
- Step 2:** In the lower panel of the Transaction Filters tab, click Add Measure.
- Step 3:** Either select the measure from the drop-down list in the Measure for Transaction Filter dialog box, or click New Measure to display the Measure dialog box, which enables you to create a new measure.
- Step 4:** In the Label textbox, enter the name of the measure as it should appear to the end user on the Momentum ticksheet.

Default Settings on Momentum Ticksheets

The transaction filter is always applied to a set of regular attribute filters. Click Add Filters in the lower panel of the tab and select those filters that comprise the set from the Filter dialog box. Only filters from the same constellation as the transaction filter are allowed.

Default Settings on Momentum Ticksheets

You can specify any configuration of Momentum filters and save this to a default ticksheet in the same way as you would in Clarity or Relevance.

Momentum has two sets of defaults that must be set for a ticksheet: the default settings for the ticksheet itself, which are set the same way as for Clarity or Relevance, and the default settings on the filter pop-up window that displays whenever you create a new filter.

To set the default settings for filters, follow these steps:

- Step 1:** Add all of the default filters that you want to use on the default ticksheet.
- Step 2:** Click the *Restrict list further* link.
- Step 3:** In the filter pop-up window, set all of the default settings.
- Step 4:** Close the window using the Cancel button. (Do not use the Close button.)
- Step 5:** Save the ticksheet as the default as you would a Clarity or Relevance ticksheet. All of the defaults, including the ones in the filter pop-up will be saved for the ticksheet.

EDITING, DELETING, AND DUPLICATING TICKSHEETS

After you have created a ticksheet, you can right-click it and select Edit from the pop-up menu to open its ticksheet dialog box for editing.

You can also use this pop-up menu to delete, duplicate, or export the ticksheet metadata file.

This completes the section on setting up a constellation. To set up another constellation, return to “Setting Up a Constellation” on page 106.

EXTRACTION

After you have configured your EpiCenter, you may use the Extraction folder to set up the extraction process described in Chapter 2. The Extraction folder consists of the sub-folders Data Stores, Extractors, Jobs, and External Tables, each of which is discussed below.

Data Stores

You can use the EpiCenter Manager's Data Store dialog box to create as many data stores as are appropriate for your site. Each extractor has a single input and output data store. Because jobs consist of multiple extractors, a job has multiple input and output data stores. For example, one job might extract data from two source systems into one EpiMart, and another job might extract data from a source system into external tables and from external tables into staging tables. In the second job, the external tables serve as both input and output data stores to different extractors.

The Data Store Dialog Box

Each data store has a Data Store dialog box (see Figure 49), and this dialog box has two tabs: General and Properties.

The General tab has two panels: Data Store Type and Data Flow. The Data Store Type panel of the General tab is where you enter the name and description for the data store and select its type: Microsoft SQL Server, Oracle, Generic ODBC Data Source, or File.

Note: The Logging and Datamart options in the top-right corner are read-only and simply indicate when the opened data store is one of the two special built-in data stores.

The Data Flow panel of the General tab has these options:

- *Allow Use as Input Data Store* allows the data store selected in the Extractor dialog box (Figure 50, on page 181) to be used as the input of an extractor.

The Data Store Dialog Box

The source systems listed in the Source System drop-down list serve to distinguish source system identifier keys that may clash between different database systems. For example, if an account has an ERP and an SFA system, each of which has a Customer table, both may contain a customer number 100. To distinguish between these two records, two different source system identifiers must be used.

Important: Fact rows will join only with dimension rows that have the same source system identifier. For this reason, unless the site actually uses two or more logical source systems, select Datamart Source from the Source System drop-down list (the default).

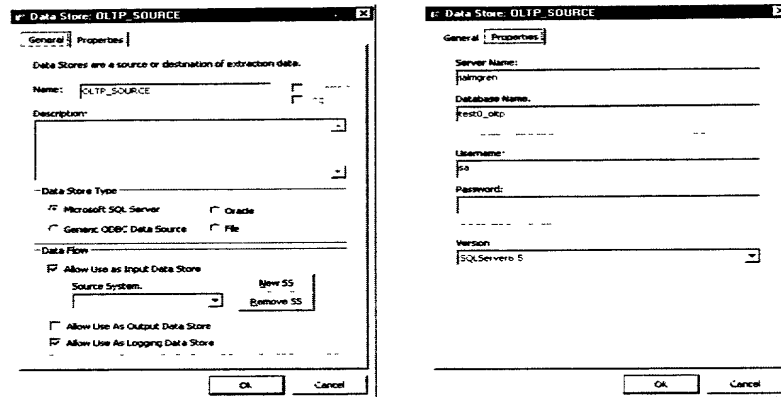


Figure 49: Data Store Dialog Box: General and Properties Tabs

- *Allow Use as Output Data Store* allows the data store selected in the Extractor dialog box (Figure 50, on page 181) to be used as the output of an extractor. De-selecting this option ensures that one does not accidentally write data to a database that is read-only; the data store name does not appear in the list of available output stores for extractors.

- *Allow Use as Logging Data Store*

Epiphany has special metadata tables for the purpose of logging its activity. Normally, this logging is written to the EpiMeta database.

Logging to the EpiMeta, however, may increase its size significantly, so you may prefer to log to another data store. You can use the *logging_mssql.sql* script (included with the Epiphany software) to create a new logging database. If you run this script on another database, then a new data store can be used to log extractions.

Note: The *Allow Use as Logging Data Store* option must be selected in the Data Store dialog box. The purpose of this option is to ensure that logging is directed to the proper logging data store.

The Properties tab has different fields, depending on the data store type. You can use the Properties tab to enter these fields:

- For SQL Server, enter the server's name, database name, username, and server's password and version.
- For Oracle, enter the SQL Net name, username, the server's password, and the version number (Oracle8).
- For a generic ODBC data store, enter the DSN name and ODBC driver name, and server administrator's username and password. (You must also set up a DSN for the data store system using the Data Source Administrator in the Windows NT Control Panel.)
- For a data store of type file (*JobLogFile*), enter its file directory path.

Extractors

Modifying the Default Data Stores

You will need to modify the default data stores, which are *EpiMart*, *JobFileLog*, and *LoggingDB*. Double-click their folders to open them and fill out the dialog boxes as follows:

- *EpiMart* specifies your EpiMart as a data store.
The only value that you can configure is the name of your EpiMart. You can use the Properties tab of the EpiMart Data Store dialog box to enter this database name, for example *Corp_EpiMart*.
- *JobFileLog* specifies the data store for EpiChannel job logs.
In the General tab, the settings are correct for the default usage. The Data Source Type is a file, and the Data Flow is set to *Allow Use as Logging Data Store*.
Click the Properties tab, and change the directory for the *JobFileLog* from CHANGE ON INSTALL to the correct directory name. A valid directory name is required for a successful extraction. Leave the file name blank.
- *LoggingDB* specifies the data store for the EpiChannel logs.
In the General tab, the Data Source Type is your server, and Data Flow specifies: *Allow Use as Input Data Store* and *Allow Use as Logging Data Store*.
You can accept the default values for the log database's file name and directory path, unless you have located the log in a different database.

Note: \$\$DEFAULT refers to the current EpiMeta.

Extractors

In general, SQL statements extract data and semantic instances merge data. The extractor SQL statements may be either stand-alone or extraction (pull/push) SQL statements. Stand-alone statements are typically used to produce a side-effect, such as creating or dropping indexes.

Extractor filters control incremental extraction through the use of special SQL macros.

To define an extractor and its filters:

- Step 1:** Choose New Extractor from the Extraction menu. The Extractor dialog box that displays has two tabs: General and Filter (see Figure 50).
- Step 2:** In the General tab, enter the extractor's name and description.
- Step 3:** Select the appropriate input and output data stores from the drop-down lists. *EpiMart* and all data stores that are enabled for input/output appear in these lists.
- Step 4:** To create a new input or output data store from this dialog box, click the New Input or New Output button, which displays a blank Data Store dialog box for editing.
- Step 5:** Click the Edit Steps button to open the Extractor Steps dialog box (see Figure 51). The buttons on the right side of this dialog box are the means by which you define a new group, new SQL, or a new semantic instance.

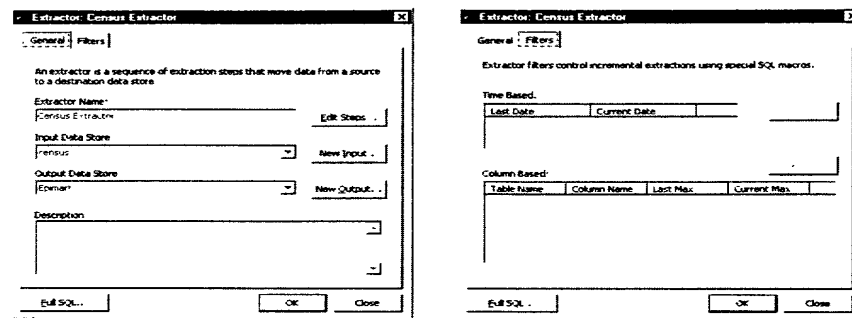


Figure 50: Extractor Dialog Box

Extractors

- Step 6:** In the Filters tab, click the Clear Filters button to clear previous date or timestamp filter memory from the system. For instance, using the `$$TIMESTAMP_RANGE` SQL macro causes the last date of extraction to be “remembered” for this extractor. (See “Extracting New Rows Only” on page 70 for more information.)
- Step 7:** The extraction filter macros are built-in. To modify a macro, click Full SQL and choose its SQL file from the dialog box. Modify the macro and save it with the same file name. This creates a text file with the post translation version of all SQL statements in the extractor (excluding semantics).

The Extractor Steps Dialog Box

Figure 51, on page 183 shows an Extractor Steps dialog box in which extractor steps have already been defined for the extractor. Select the extractor from the drop-down list, or click New to display the Extractor dialog box (Figure 50) in which you can create a new Extractor.

All of the EpiCenter’s extraction groups, which consist of extraction steps, appear in the right side of the dialog box.

When working with the Extractor Steps dialog box, keep these definitions in mind:

- A *job* consists of extractors, which can be shared between jobs. (A job also consists of system calls.)
- An *extractor* consists of extraction groups, which can be shared between extractors.
- An *extraction group* consists of extraction steps that are either SQL statements or semantic instances.

When extraction groups have been defined, you may select the ones for this extractor and move them to the Extractor Steps listbox on the left. Use the arrows to move them over, and the Up and Down buttons to order them in the list.

Because an extractor step must be enabled (checked) to be used in a job, de-select any extractor step that you do not want to be run. You may de-select an extractor step to restart a job that failed, or to debug a subset of the steps. In most cases, if a job fails, you should re-run the entire job.

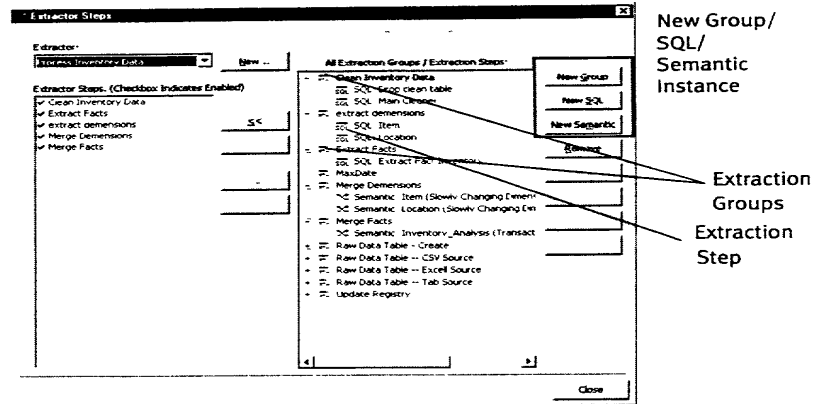


Figure 51: Extractor Steps Dialog Box

Notes:

The *End of Extraction* extractor is provided by default. See “External Tables” on page 50 for a description.

You may right-click a group or an extraction step to display a contextual pop-up menu (for adding a New Group, New SQL, and so forth).

09625518.072500

Extractors

Extraction Group

To define an extraction group:

Step 1: Using the Extractor Steps dialog box, click New Group.

Step 2: Enter the group name in the dialog box; for example, *Order Raw* to extract raw data from the Order master table on the source system.

This name appears in the All Extraction Groups/Extraction Steps list in alphabetical order.

Extraction Steps for the Group

Next, you need to define the extraction steps for the group. This group can contain any number or combination of SQL statements or semantic instances. If the step requires SQL, EpiCenter Manager provides a template for sample SQL.

To define SQL for one of the steps in this group:

Step 1: Select its group name in the All Extraction Groups/Extraction Steps listbox, and click New SQL.

The SQL Statement dialog box is displayed (see Figure 52). In the upper panel are the General and Store Types tabs; the SQL and Description tabs are in the lower panel. (You can use the description tab to add a description for your reference.) You can resize this window.

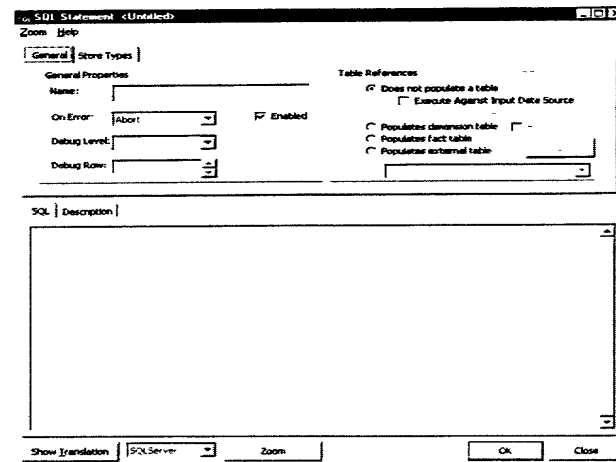


Figure 52: SQL Statement Dialog Box

- Step 2:** In the General Properties panel of the dialog box, enter the step's name (as it will appear in the All Extraction Groups/Extraction Steps listbox). For example, a step named *Customer Raw* extracts raw customer data.
- Step 3:** You can define a step but not have it execute by de-selecting Enabled.
- Step 4:** Select the action to be taken if there is an error with the step. The default is to abort the step. (You can also ignore the error, or have it print to a log and continue.)

Extractors

Step 5: Select the debug level. These levels correspond to the verbosity levels on the **extract** command line. See “EpiChannel Debugging Levels” on page 78.

Step 6: Select the row number at which the debug level you selected above should go into effect, if applicable. If you leave this field blank, the debug level will take effect immediately.

Step 7: In the Table References panel of the SQL Statement dialog box, select whether the step:

- does *not* populate a table (and if this is true, select whether it executes against input data store, or retains its default behavior of executing against the destination data store);

—or—

- populates one of the following types of tables: a dimension table, a fact table, or an external table.

Most SQL statements are used to populate staging tables (or sometimes, external tables). A SQL statement may, however, be used to achieve a side-effect, in which case you will set it to “not populate a table.” In this case, the statement is executed, and any returned results discarded.

Step 8: If the step references a table, select the table from the drop-down list.

Step 9: Execute Only. (*This option is not supported for this release.*)

Select this when SQL is to be expanded to reflect the structure of one of these tables, but the returned rows from the SQL statement are not meaningful. EpiChannel will not insert rows in the statement.

Extraction SQL is executed against the input data store and must be in the dialect of that database engine. The results are stored in the destination data store. EpiChannel automatically resolves SQL dialect problems if you use the Epiphany database-vendor independent macros consistently in your SQL. (See Appendix A, “Epiphany Macros” for more information.)

Display Sample SQL

If the step populates a table, clicking the Template button displays sample SQL for the step in the lower panel of the SQL Statement dialog box. The SQL needs to be modified to contain the FROM and WHERE clauses appropriate for the tables in the source system, but the template lists the columns that must be returned. It does not matter in what order the columns are returned as long as they have the proper column names (and “extra” unused columns may be returned).

You can have the system show the how the macros will be translated for your server. Click the Show Translation button and select your server type from the drop-down list.

The SQL Results window displays the SQL in color-coded text. The main menu commands enable you to use standard edit commands to search and replace lines, and you can set bookmarks on lines. The Tools menu has Undo and Print commands, and commands for clearing text with and without clearing the buffer.

The SQL Statement dialog box is re-sizable. You can also click the Zoom button to have the text area fill up the entire SQL or Description window.

Restrict Data Stores for the Extraction

To restrict the input/output data stores for this extraction:

Step 1: Specify them in the Store Types tab of the SQL Statement dialog box.

Step 2: Click Add Type to add a type.

If a type is listed, the statement is disabled for any other type except those listed. For example, you could use the same SQL twice, once with Microsoft SQL Server syntax and once with Oracle syntax, and have one or the other statements automatically disable itself when the source database is of the wrong type. This is of value when the group that contains the SQL statement is shared by multiple extractors. Another approach to handling SQL dialect problems is to use Epiphany SQL replacement macros instead of database vendor-specific constructs. See Appendix A, “Epiphany Macros” for more information.

Extractors

Additional Steps

To create additional steps within the same group, click OK to return to the Extractor Steps dialog box, and repeat the appropriate steps as described above.

Semantic Instance

To set up a semantic instance:

- Step 1:** Either create a new group or select an existing group in the Extractor Steps dialog box.
- Step 2:** Click the New Semantic button. The Semantic Instance dialog box is displayed.
- Step 3:** Select the fact or dimension table that the semantic instance references.
- Step 4:** Select the Object from the drop-down list. An object is either a fact table or a base dimension table (the one to be operated on by the semantic type).

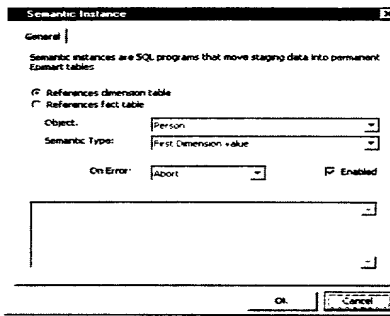


Figure 53: Semantic Instance Dialog Box

- Step 5:** Select the semantic type from the drop-down list. See Appendix F, "Semantic Types" for descriptions of semantic types.

- Step 6:** Select the action to be taken upon error: abort, ignore, or print. Enabled must be selected for this action to take place.
- Step 7:** To create additional semantic instances for this group, repeat Steps 2 through 4.

Jobs

The Job dialog box enables you to order the job steps (extractors and system calls) that comprise a job. The Add Job Steps dialog box (available through the Job dialog box) enables you to create system calls.

An EpiCenter has a default job consisting of these default job steps: the Aggbuilder system call for aggregation, the End of Extraction extractor, and the Refresh system call for refreshing the Application Server. (For the Refresh system call to work, the Application Server must be configured as described in the *Epiphany Installation Guide*.) If Momentum is installed, the MomentumBuilder system call is also included as a default job step.

You may customize the default job and create as many other jobs as necessary.

To open the default Job dialog box, double-click it. To open a new job dialog box, right-click the Jobs folder and select New Job.

Jobs

The Job dialog box has three tabs: General, Job Steps, and Store Roles (Figure 54).

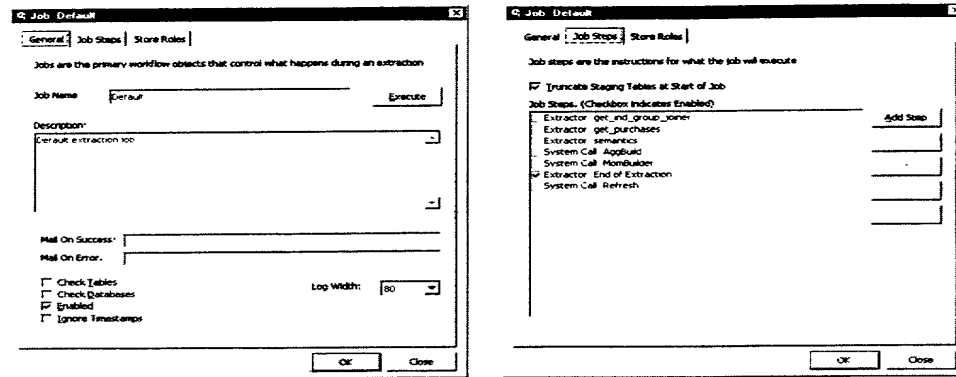


Figure 54: Job Dialog Box: General and Job Steps Tabs

The General tab provides options you can set that control the following aspects of a job:

- For a job other than the default one, assign a name and description.
- Enable a job. EpiChannel executes enabled jobs only.
New jobs are initially enabled. You may, however, disable a completely functional job in some circumstances to accommodate system changes. For example, if a database is in the process of being moved or repaired, jobs that populate that database could be disabled as a protection against accidental execution.
- Select the width of the log, either 80 or 132 columns. Select 80 for VGA monitors and 132 for SVGA monitors.
- Assign addresses for e-mail notification of the job's success or failure. See "Configuring E-Mail" on page 196.

- Request that before executing the job, EpiChannel check that all databases referenced by any job databases and all EpiMart tables are available. Unless you are sure that this is true, select this option.
- Request that all timestamps be ignored during job execution (Ignore Timestamp).

For example, by ignoring timestamps, you could retrieve all rows, without any date filters (based on EpiChannel's special filtering used for incremental extractions; see the Filters tab of the Extractor dialog box (Figure 50, on page 181)).

- Turn off the truncation of staging tables. (Use this only in development mode to restart a job without truncating any tables.)
- Execute a single job.

Click the Execute button in the Job dialog box (Figure 54) to execute this job. You can use the Execute Job dialog box (Figure 55) to do the following:

- Run the job as a trial run.
- Normally, the proper instance name is already selected. You can enter a new one by clicking the New button.
- Select a debug level that corresponds to the EpiChannel (**extract**) verbosity levels (although you cannot specify row numbers). This debug level takes effect before the first SQL statement.
- Indicate the total number of rows to be transferred in a pull/push operation: all rows or the first N rows. This value is reset with each extraction statement.

Warning: Truncation is enabled for this job.

Adding Extractors and System Calls as Job Steps

Using the Execute button is equivalent to invoking the **extract** command with this job as the job option. The Execute button, however, also supplies the database name, password, and so forth, if necessary (no instance is supplied) so that **extract** does not depend upon the Registry entries.

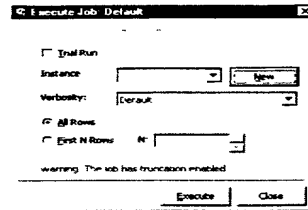


Figure 55: Execute Job Dialog Box: Selecting Debug Level

You can use the Store Roles tab to set up the data stores for the job (by default *EpiMart* and *JobFileLog*) and indicate their roles; for example, *working directory* or *log*. See “EpiChannel Output” on page 80. A *working directory* is required.

You can use the Job Steps tab to do the following:

- Enable job steps by selecting them in the check box to the left of their names. Only enabled job steps are run.
- Change the order of job steps, using the Up, Down, and Remove buttons.
- Update the job definitions in metadata when you modify the job steps. (Click Edit to open another dialog box in which you can update a job step.)

Adding Extractors and System Calls as Job Steps

You will also use the Job Steps dialog box to add one of your defined extractors as a job step and to create system calls. Click the Add Step button, which displays the dialog box shown in Figure 56.

Adding Extractors and System Calls as Job Steps

To add an extractor:

Step 1: Select Extractor.

Step 2: Select the extractor from the drop-down list, and click OK.

Click New to display the Extractor dialog box (Figure 50) in which you can create a new extractor.

Step 3: The extractor now appears in the Job Steps list.

As mentioned, sites with more complex databases may require multi-stages and additional commands, such as lookup tables, aggregation splits, gathering data into ranges (binning), and duplicate detection. For this purpose, you may use system calls, which are executed during a job as if invoked from the console's DOS command line.

To create a system call:

Step 1: Select System Call.

Step 2: Enter the name of the system call, the action to be taken upon error (abort, ignore, or print), the command line, and an optional description.

Step 3: Click OK to add the system call as a step.

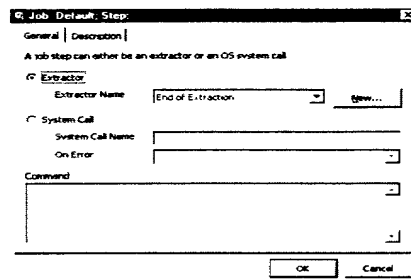


Figure 56: Add Job Step Dialog Box

External Tables

External Tables

Extraction statements are sometimes directed to load external tables that serve as intermediary tables for multi-staged extraction. Epiphany supplies one external table called *last_extract_date*. Use of this table is recommended, but optional. See “External Tables” on page 50.

To define external tables and their columns:

- Step 1:** Choose New External Table from the Extraction menu.
- Step 2:** Enter the name and any description for the table in the External Tables dialog box (see Figure 57).
- Step 3:** By default, the data in the external tables is deleted (truncated) after extraction. In most cases, you will accept this default.

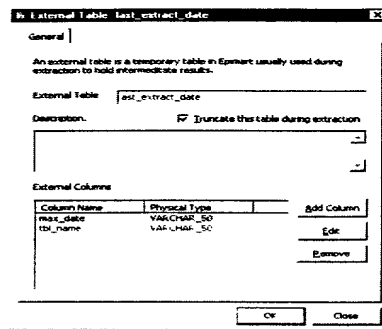


Figure 57: External Tables Dialog Box

To add a column:

- Step 1:** Click Add Column and enter the name in the External Column dialog box.

Momentum Extraction

Step 2: Select the physical type. See Appendix D, “Physical Type Values” for descriptions of these physical types.

Add as many columns as necessary. After you click OK in this dialog box, the columns are added to the Column Name list in the External Tables dialog box.

Momentum Extraction

To run the Momentum application, use the built-in MomentumBuilder job (see Figure 58) in which MomentumBuilder is a system call job step. The order of the extraction is as follows:

Step 1: SQL/Semantics

Step 2: AggBuilder

Step 3: MomentumBuilder

Step 4: End of Extraction

For information about running MomentumBuilder as a standalone executable and to determine if MomentumBuilder extracted the correct data, see “MomentumBuilder” on page 60.

09625518 072500

Configuring E-Mail

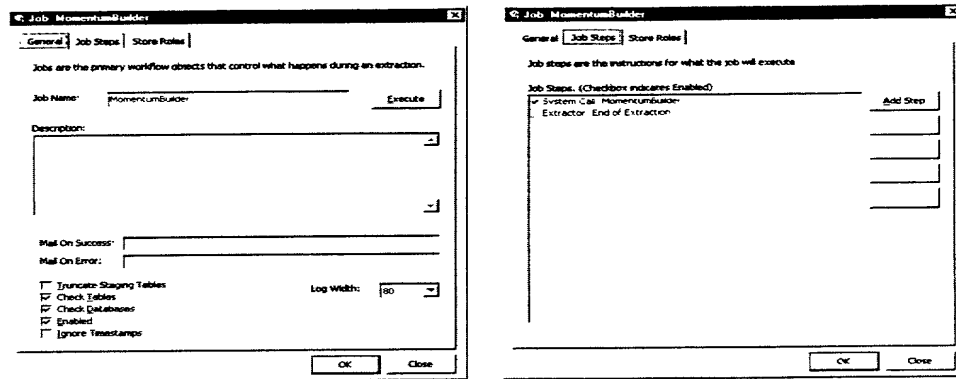


Figure 58: Momentum Job Dialog Box

Configuring E-Mail

To have the system automatically send e-mail notification of the outcome of a job (either successful completion or job failure), EpiCenter Manager needs to know your e-mail Profile name and password. These must match the information in the Mail and Fax Control Panel. Windows uses this Control Panel to define the profile of mail accounts, where each profile could tie users to different mail servers. The operating system then makes mail to this server possible via the SMAPI program interface used by EpiChannel.

To determine your mail profile name, follow these steps:

Step 1: Open the Mail and Fax Control Panel (from the Start menu, choose Settings\Control Panel and double-click Mail and Fax).

You may be given a list of profiles or may be placed into the only profile available. Either remember your profile name or click Show Profiles to display the name of the profile that was automatically opened.

09625518.072500

Configuring E-Mail

Step 2: Choose Configuration from the EpiCenter menu.

The list of EpiCenter configuration variables includes the Mail Password and Mail Profile Name.

Step 3: Click Mail Profile Name and enter the name of the profile as shown in the Mail and Fax Control Panel in the Value textbox. Click Update.

Step 4: Click Mail Password and enter the password. Click Update.

The mail password depends upon which mail server is used and follows the rules of this server. Often, the password will be the same as the Windows NT user password.

Note: Because this password may be visible to others in mail logs or in metadata exports, do not use a password of any importance. Create a new mail user on your mail server exclusively for EpiChannel if revealing this password presents a problem.

Verifying that E-mail Notification Works

To test if e-mail is configured correctly, follow these steps:

Step 1: Open a Job dialog box for a job that simply runs and exits. In the General tab, enter the addresses for mail on success and mail on failure.

Step 2: Run the job.

If you receive e-mail notification, you have set up EpiChannel e-mail correctly. If not, repeat the steps for configuring e-mail given above and run another job. Please contact Epiphany Customer Support if there is still a problem.

Configuring Outlook Exchange for EpiChannel E-mail Notification

Follow these steps to configure Microsoft Outlook Exchange 98 for EpiChannel e-mail notification:

Step 1: Install Microsoft Outlook Exchange 98 according to the instructions on the screen. When prompted to select which kind of installation, select Corporate/Workgroup. Reboot your computer.

Truncating Tables

Step 2: Start Outlook Exchange, and choose to configure an Internet Mail account.

Step 3: When prompted, enter a Profile Name for this account.

Note: The Mail Profile in the EpiCenter Manager Configuration dialog box is the default e-mail address for the EpiCenter Manager user.

Step 4: Set up the account with an appropriate e-mail account name, user name, and organization.

Step 5: For the e-mail and reply address, enter a bogus address, such as *foo@bar.com*.

Step 6: In the Servers tab, set both the incoming and outgoing mail servers to the name of the mail server that can route mail outside the company's firewall. (This name is not case-sensitive.) In the Incoming Server section, enter any account name and password. (Because EpiChannel never uses Outlook Exchange to check incoming mail, these can be any values.) Accept the default values for the other tabs.

Step 7: Exit Outlook Exchange.

Note: If the mail server is installed on a UNIX platform, you can test it by logging on with a shell account and sending mail to yourself and Epiphany from the UNIX mail program.

Truncating Tables

By default, all staging tables and external tables are truncated prior to an extraction. You can set truncation on or off via individual fact, dimension, and external table dialog boxes, or use one dialog box to define all tables in a constellation. They both refer to the same fields in the database.

To define the set of tables for truncation:

Step 1: Choose Edit Truncation from the Extraction menu, or click the Edit Truncation toolbar icon in the toolbar menu.

Step 2: In the Truncate Tables before Extraction dialog box (Figure 59), select tables to be truncated.

All of your tables are listed in columns by type (Dimensions, Facts, and External tables). Check each one in the list that you want truncated. (Clicking the All button selects all of the tables in a column; clicking None de-selects all of them.)

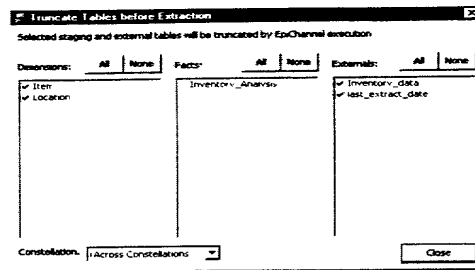


Figure 59: Truncate Tables before Extraction Dialog Box

Purging EpiMart Tables

You can remove tables from the EpiMart that are no longer needed. For example, you may have unused tables to delete or, as a result of metadata changes, have decreased the number of aggregates. These higher numbered aggregate tables remain in effect until you purge the EpiMart tables.

You should first try a trial run.

To purge database tables:

Step 1: Choose Purge EpiMart Tables from the EpiCenter menu.

Step 2: Select Trial Run in the Purge tab, and click Go.

Security

After the run, click the Results tab to see which tables will be deleted. If these results are acceptable, return to the Purge tab, de-select Trial Run, and click Go to delete these tables from your EpiMart.

SECURITY

The Epiphany system provides two areas of security:

- *Authentication*, or a user's ability to log on to the system. Authentication is determined by the Windows NT operating system.
- *Access rights*, or the permissions a user has after logging on. Access rights include the user's ability—
 - to open a ticksheet.
 - to save queries for those ticksheets. Saved queries (called *reports*) are lists of the options that a user selected on a ticksheet (to generate a report).
 - to access data based on the values of dimensional attributes; that is, to restrict access rights to the ticksheet to certain dimensional attribute values. You can use this kind of security to allow some users to see data for one region but not another.

The Epiphany system can also use Windows NT groups to administer access rights.

You will use the Security folder in the EpiCenter Manager directory tree to add groups and users to the system. The two work in tandem: groups must have users, and users belong to groups. When setting up a new Epiphany system, first set up groups and then add members after you have set up users. (Windows NT groups members can simply be imported.)

The Security folder also contains the Report Gallery, which enables the administrator to organize all saved reports in folders for groups and users. See "Report Gallery" on page 209 for more information.

Right-clicking the Security folder display a pop-up menu with commands for New Group, New User, Report Gallery, Import NT Group, Import NT User, and Export.

Setting Up Groups

Members of a group share similar access rights, or permissions. In general, you should attach these to groups instead of users to simplify maintenance. Access rights set for a user take precedence over access rights set for groups to which the user belongs.

To define a group, right-click the Group folder and select New Group from the pop-up menu. (This menu also has commands for exporting all groups, importing NT Groups, and refreshing the group folder list.) Complete the information requested in the Group Definition dialog box tabs (General, Membership, Ticketsheet Access, Column Access, and Description) as described in this section. (Use the Description tab to document the group for your own reference.)

Step 1: In the General tab (Figure 60), enter the group's name.

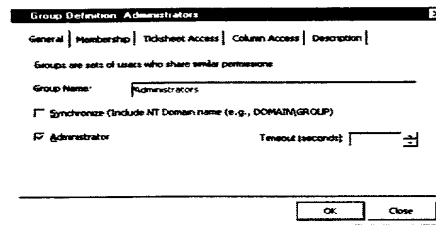


Figure 60: Group Definition Dialog Box: General Tab

Step 2: Select the Synchronize option to add new users to the group in an “autopilot” fashion via Windows NT synchronized groups. You need to precede Windows NT synchronized group names with their Windows NT domain name prefix (and matching name).

When the synchronize option is set, each time a user logs in, the Windows NT security API is accessed for a list of group names to which that user belongs. If the user is a member of an NT group for which there is no membership record in EpiCenter Manager, then a new record is

Setting Up Groups

created. Conversely, if the user is no longer a member of an NT group, but there is such a membership record in EpiCenter, then the latter record is removed.

- Step 3:** Select Administrator if members of this group have administrative rights. An Administrator can see all ticksheets and reports in the system, but cannot modify special folders, such as the Public folder. (See “Report Gallery” on page 209 for a discussion of Public folder.)
- Step 4:** Enter the seconds for the maximum time that a query will run for members of this group. As an administrator, you may set a time-out limit to ensure that someone does not monopolize database engine resources.
- Step 5:** You can use the Membership tab (Figure 61) to add already created users to the group. Users may be members of multiple groups. Click Add User.

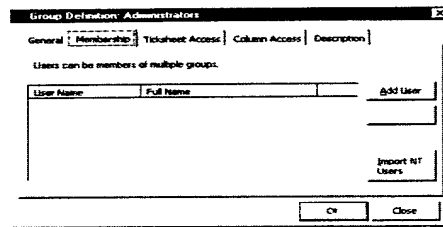


Figure 61: Group Definition Dialog Box: Membership Tab

- Step 6:** Select the user's name from the Choose User dialog box and click OK, or click New, which displays the User dialog box.
- Step 7:** To import NT users into the group, click Import NT Users and enter the domain in the dialog box.
- Step 8:** Select one or more members from the Choose NT Users dialog box.

09625518.072500

Setting Up Groups

Step 9: You can use the Ticksheet Access tab to assign all members of the group access to certain ticksheets. Before Epiphany users can open any ticksheet, they must be granted access to it. Users have the ability to see and use all ticksheets granted access to their groups by the administrator. Click Add Ticksheet and select ticksheets from the Choose Ticksheet dialog box. The dataset that the ticksheet belongs to and ticksheet type are displayed in the tab.

Step 10: When a user clicks the Create Report button in a ticksheet to generate a report, some results may be filtered automatically so that the report's output is restricted.

You can use the Column Access tab (Figure 62), which is the same for users and groups, to restrict access rights for the ticksheet to certain attributes (dimension columns).

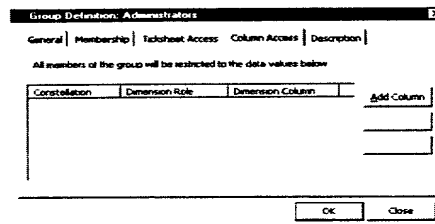


Figure 62: Group Definition Dialog Box: Column Access

Setting Up Groups

Click the Add Column button, which displays the Dimension Column Access Group Definition dialog box (Figure 63).

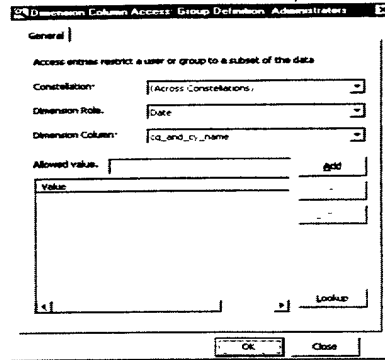


Figure 63: Dimension Column Access Group Definition Dialog Box

Step 11: Select the Constellation to which the dimension column belongs from the drop-down list. You may also apply this restriction across all constellations (for example, apply the date dimension).

Each ticksheet is associated with a constellation. When a user runs a report, only those dimension column restrictions that apply to that constellation are applied. Thus if an EpiCenter has two constellations, one for Sales and one for Expense, there will be no filtering on Expense dimensions when a user works with a Sales ticksheet (unless Across Constellations has been selected).

Step 12: Select the Dimension Role for this dimension column from the drop-down list.

Step 13: To display all of the values in the dimension column, click Lookup, which directly accesses the EpiMart data. Select a value or values from the listing and click OK to add it to the value list.

Setting Up Users

The values entered in this list should correspond to actual database values in the base dimension table to which that dimension column corresponds. For example, selecting *Date.fy_name* as the field with the values 1997 and 1998 causes all reports to be filtered with these values.

Step 14: If you know the exact value of the dimension column whose values will be accessible to the group, type it in the Allowed values textbox, and click Add to place it in the Value listing below. Repeat this step to add additional values.

Step 15: Click OK to return to the Group Definition tab.

After a Group has been defined, it becomes a subfolder of the Group folder. You can right-click a group folder and use the pop-up menu to set up a new group, or to edit, delete, or duplicate the group. A duplicate group is identical except for its group name.

Setting Up Users

All authorized users of the Epiphany system will appear as icons in the Users subfolder of the Security folder. By default, new users have these permissions:

- no access to any ticksheets.
- no group membership.
- inability to save any reports.
- access to all data in all dimensions—there are no dimensional attribute restrictions.

The access rights set for a user have precedence over access rights set for groups to which the user belongs.

To set up a new user:

Step 1: Right-click the Security folder and select New User. The User dialog box (Figure 64) that displays has tabs for General, Membership, Ticksheet Access, and Column Access.

Setting Up Users

Step 2: You can use the General tab to enter the person's username and first and last names.

The first and last names are used for display purposes only. If they are missing, then the username is displayed.

For Windows NT authentication, enter users with their Windows NT domain prefix to distinguish among users with the same name but who are in different domains.

Step 3: Enter the seconds for the maximum time that a query will run for this user. As an administrator, you may set a time-out limit to ensure that a user does not monopolize database engine resources.

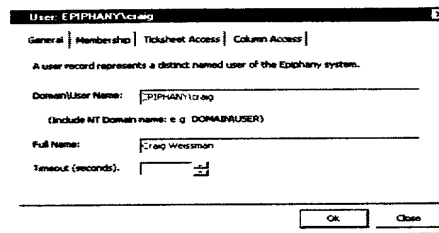


Figure 64: User Dialog Box: General Tab

Step 4: After groups have been created as described in "Setting Up Groups" on page 201, you will use the Membership tab to assign group memberships to the user.

Click Add Group and select one or more groups from the Choose Groups dialog box. (Hold down the Shift key to select more than one.) A user can be a member of multiple groups, but for security reasons needs to be a member of a single group (called the primary group) when running queries. After assigning group memberships, select the main group in the list and then select the Primary option in the tab.

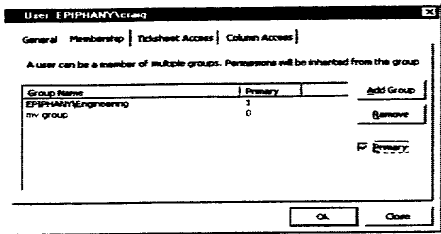


Figure 65: User Dialog Box: Membership Tab

- Step 5:** Before Epiphany users can open any ticksheet, they must be granted access to it. Users have the ability to see and use all ticksheets to which they have access, and to all ticksheets granted access to their groups. In the Ticksheet Access tab, click Add Ticksheet and select the ticksheets in the list that the user may access. Hold down the Shift key to make multiple selections.
- Step 6:** You can use the Column Access tab (Figure 66) to restrict access rights for the ticksheet to certain attributes (dimension columns). Click Add Column, which opens the Add Column dialog box.

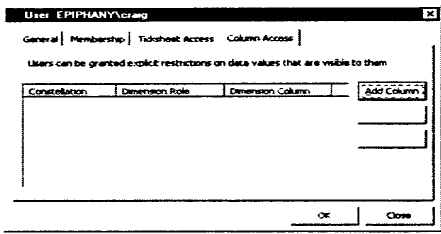


Figure 66: User Dialog Box: Ticksheet Column Access Tab

Setting Up Users

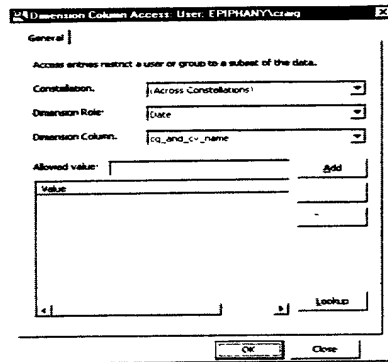


Figure 67: Dimension Column Access Dialog Box

- Step 7:** Select the Constellation to which the dimension column belongs from the drop-down list. You may also apply this restriction across constellations. Each ticksheet is associated with a constellation. When a user runs a report, only those dimension column restrictions that apply to that constellation are applied. Thus if an EpiCenter has two constellations, one for Sales and one for Expense, there will be no filtering on Expense dimensions when a user works with a Sales ticksheet (unless you have selected Across Constellations).
- Step 8:** Select the Dimension Role for this dimension column from the drop-down list.
- Step 9:** To display all of the values in the dimension column, click Lookup, which directly accesses the EpiMart data. Select a value or values from the listing and click OK to add it to the value list.

The values entered in this list should correspond to actual database values in the base dimension table to which that dimension column corresponds. For example, selecting *Date.fy_name* as the field with the values 1997 and 1998 causes all reports to be filtered with these values.

Step 10: If you know the exact value of the dimension column whose values will be accessible to the group, type it in the Allowed values textbox, and click Add to place it in the Value listing below. Repeat this step to add additional values.

Step 11: Click OK to return to the User dialog box.

Report Gallery

Note: The Report Gallery in EpiCenter Manager is for administrative use only. Many of the features of this Report Gallery can also be accomplished by end users using the Report Gallery in the Web interface.

You can use EpiCenter Manager to browse the saved reports for a ticksheet, as well as view who has access to those reports. All of the saved reports for the Epiphany system are organized in folders in the Report Gallery. There are folders for users and groups, and a Public folder. As mentioned, whenever a new ticksheet is created, the administrator should save one report in the Default subfolder in the Report Gallery's Public folder (see "Report Gallery" on page 209) and make it the default.

To display the Report Gallery, right-click the Security folder and select Report Gallery from the pop-up menu. The Report Gallery (see Figure 68) is displayed. The top panel shows the folders organized in a tree hierarchy, and the lower panel lists the reports for a selected folder by report name, report type, ticksheet name, date last modified, and the person who modified it.

Report Gallery

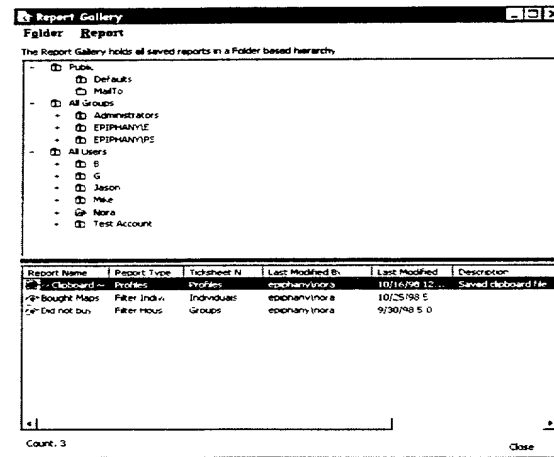


Figure 68: Report Gallery in EpiCenter Manager

Each group and user has a default report folder called Defaults, which is located in the user's or group's folder. There is also a Public Defaults folder.

There is a default report for each ticksheet at the user and group levels. A default report is the set of ticksheet settings that a user sees when he or she first opens that ticksheet. In general, the most specific default report is used. Thus if a user-level default report exists, that report is opened in the user's Web browser. If no user report exists but a group default report exists for the ticksheet, the group default report is displayed. The default report in the Public folder is available if there is no specific user or group default report.

When a user saves a report, it is saved in whatever folder the user is currently in. This can be the user's default folder, a folder for which the user has write access, or the Public folder.

The Report Gallery's main menu has menus for Folder and Report, which are described below.

Folder Menu

Note: Right-clicking in the top panel of the Report Gallery displays a pop-up menu with the Folder commands.

You can use the Folder menu to—

- create a folder for saved reports. Choose New from the Folder menu, select the folder or subfolder, and enter the new folder name in the dialog box.
- delete a selected folder.
- display the folder's properties. (You can also double-click the leaf folders to open the Folder dialog box.)

The Folder dialog box has two tabs: General and Permissions. The General tab shows the folder's name, path, date last modified, and any description.

The Permissions tab allows you to set controls for who can view and alter the folder. See "Setting Permissions" on page 212 for more information.

- find a report in one of the folders.
Enter the file name in the Find Report dialog box. You can refine your selection to report types and date modification ranges. Pressing the F3 key locates the next report that meets this criteria.
- Move folders by dragging them.

Report Menu

Note: Selecting a report file in the lower panel displays a pop-up menu with the Report commands.

You can use the Report menu to copy and move reports and folders. (This functionality is not available through the Web interface Report Gallery.)

To copy a report:

Select it, and choose Copy from the Report menu. Select the folder in which you would like to place the copy of the report, and select Paste.

You can also cut and paste a report, or explicitly delete it.

Report Gallery

Double-clicking a report opens the Report dialog box (Figure 69).

- The General tab shows the report's name, folder path, report type, ticksheet name, date last modified, and any description.
- The Permissions tab allows you to set controls for who can view and alter the report. See "Setting Permissions" for more information.

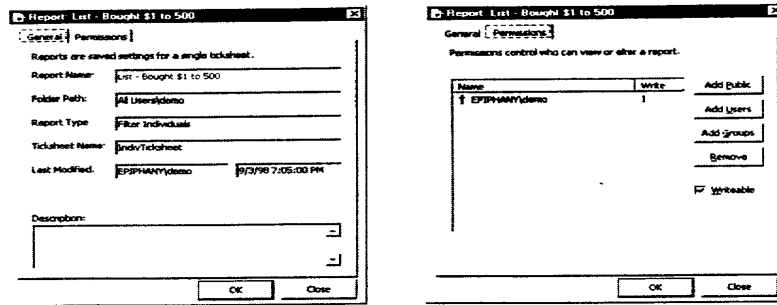


Figure 69: Report Dialog Box

Setting Permissions

You can use the Permissions tab of the Folder and Report dialog boxes (see Figure 69) to assign users and groups permission to view a folder or report. If you select Writable in the Report Permissions tab before clicking the Add button, then the users or groups you add may alter the report. Selecting Writable in the Folder's Permissions tab gives write access to any report in the folder for the user or group (although a user cannot create a new report in a folder unless he or she has write access to that folder).

- To allow all users the ability to view this folder or report, click Add Public.
- To give users access to folders/reports, click Add Users and select users from the Choose dialog box.
- To give groups access to folders/reports, click Add Groups and select groups from the dialog box.

SHARED INTERFACE

The Shared Interface folder has the subfolders Datasets and Glossary Entries, which are user-interface features shared by the ticksheets.

As part of building a ticksheet, you added Glossary entries to describe ticksheet terms to users. These terms are hyperlinked to a glossary page. When a user clicks a link, a glossary page displays which defines it. See “Setting Up a Glossary Entry” on page 134 for instructions.

A dataset is a logical grouping of similar ticksheets within a constellation. Instructions for setting up datasets follow.

Setting Up Datasets

You can group related ticksheets by dataset. In the browser window, these group names, or labels, appear in the main menu (left margin of the ticksheet) under the Dataset heading. For example, one dataset may contain Executive Summary ticksheets and another Shipment-related ticksheets.

A dataset may contain one and one only of each ticksheet type. For example, the Executive Summary dataset can have only one Clarity Table & Charts ticksheet, one Relevance Best & Worst, one Relevance Influence, one Relevance Lifecycles, and so forth.

You can use the Dataset dialog box (Figure 70) to create and describe these logical data views. When you build a ticksheet, you will assign it to a dataset (see “Assigning the Ticksheet to a Dataset” on page 127). Right-click the Dataset folder and select New Dataset. Enter the dataset name and the label name (the name as it should appear on the ticksheet). Enter any help text that will be displayed to end users.

Note: Most top-level entries in the EpiMeta need a unique name so that they can be identified for export/import purposes. The dataset name is internal to EpiMeta; users see only the label.

Generating Schema

The Ticksheet tab lists the ticksheets in the dataset. Dataset ordering applies after you have created more than one dataset. To rearrange the order in which the Datasets appear to end users, right-click a dataset in the Shared Interface folder, and select Up or Down from the pop-up menu.

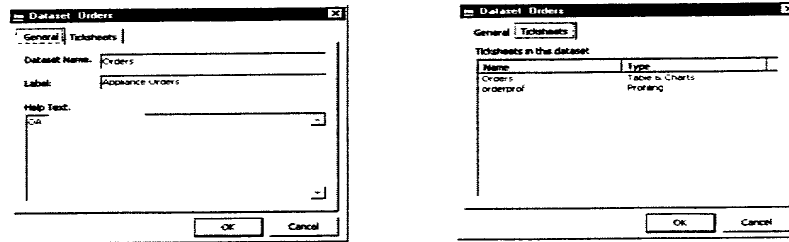


Figure 70: Dataset Dialog Box

GENERATING SCHEMA

After defining your metadata via EpiCenter Manager, you need to convert these definitions into actual tables (the EpiMart). You can use the Generate Schema dialog box to build the EpiMart tables, as well as to populate the physical date dimension table. The Date dimension table is a special base dimension table supplied by Epiphany for storing all attributes related to time. All fact tables in an EpiCenter receive a foreign key (called *date_key*) to this table.

Momentum users can use this dialog box to build the sampling table. The sampling table is a list of random numbers that efficiently produce statistically accurate samples. You can use the General tab of the Configuration dialog box to set the compression ratio for Momentum sampling (see “General Settings” on page 285).

To generate the schema for your EpiMart:

- Step 1:** Choose Generate Schema from the EpiCenter menu. The Generating EpiMart Schema dialog box has two tabs: Schema and Results.

Generating Schema

- Step 2:** In the upper panel of the Schema tab, select Trial Run to see what the results will be without making permanent changes to the files.
- Step 3:** By default, only tables that have changed since the last time you generated schema are rebuilt. Select Force Rebuild of all Tables only if you want to rebuild *all* of your tables.
- Step 4:** Momentum users have the option of selecting Build Sampling Table. A sampling table is a table that contains a statistical sample of the data.
- Step 5:** The date dimension table must be populated the first time you generate schema. Follow the instructions in “Populating the Date Dimension Table” below.
- Step 6:** Click Go (and watch the progress bar).
- Step 7:** After the trial run is finished, click the Results tab to view which tables were updated.
- Step 8:** If the results are acceptable, de-select Trial Run in the Schema tab, and click Go.

After the schema has been generated, the system records the current state of the metadata so that the next time the schema is updated, the current definitions can be compared with the new ones, and the appropriate tables can be created or altered as necessary.

Populating the Date Dimension Table

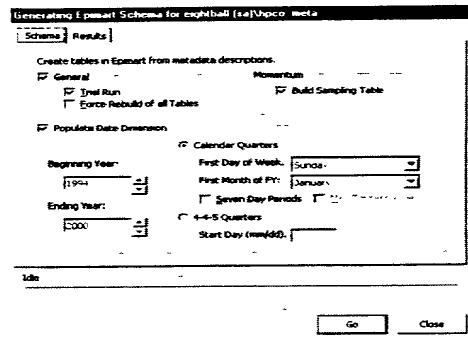


Figure 71: Generating EpiMart Schema Dialog Box

Populating the Date Dimension Table

To populate the date dimension table:

- Step 1:** Select Populate Date Dimension in the Schema tab of the Generating EpiMart Schema dialog box. (You can also choose Populate Date Dimension from the EpiCenter menu to populate the dates only; see “Populating the Date Dimension” on page 91.)
- Step 2:** Enter the values for the beginning and ending years of the EpiMart. The date range of the EpiMart should be at least as large as any dates that are found in the data you will be extracting. These values are defined in the Configuration dialog box (choose Configuration from the EpiCenter menu to open it).

For users to obtain the best results when forecasting trends using Relevance, you need to build the date dimension as far into the future as you plan to forecast. (Currently, the maximum prediction is three years

past the last date that has recorded data.) If the date dimension is not built out far enough, the user will receive columns with names such as *Dec 1999*, *Second Next*, *Third Next* instead of *Dec 1999*, *Dec 2000*, *Dec 2001*.

- Step 3:** Choose the appropriate quarter system: Calendar Quarters (three month divisions of the year) or Quarters 4-4-5. *4-4-5 Quarters* represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.
- Step 4:** *For Calendar Quarters*, enter the first day of the week and the first month of the fiscal year. Select Seven Day Periods to ensure that all fields that count up in number (such as *week_number_fq*) begin with seven day periods (that is, 11111112222222333333...13 or 14), whereas all fields that count down in number (such as *week_number_til_end_gq*) end with seven day periods (that is 14 or 13...3333333222222111111). The alternative is that weeks may overlap quarter boundaries incompletely.
- Step 5:** *For Calendar Quarters*, select Max Thirteen Week to prevent a “14th” week or “53rd” year; the 13th or 52nd weeks are simply extended as needed.
- Step 6:** *For 4-4-5 Quarters*, enter the start date of the first quarter.
- Appendix , “Date Dimension Fields” describes the date dimension fields used by the Epiphany system.

EXPORTING/IMPORTING METADATA

Epiphany provides a metadata export/import utility for moving metadata between EpiCenters and for backing up the definition of an EpiCenter. To use this tool properly, you need to understand the metadata concepts (see “Metadata Overview” on page 323).

Exporting/Importing Metadata

The export operation produces a Microsoft Access database that contains a representation of the metadata that was chosen for export. Upon import, this representation is converted back into valid Epiphany metadata in the target EpiMeta. Using EpiCenter Manager, you have granular control over which metadata objects get exported during each operation. For example, you can use the same constellation definitions, extractors, and ticksheets in several EpiCenters. Also, during import, you control whether or not to overwrite existing metadata that conflicts with what is in the import file.

Reasons for exporting files other than publishing them for import are to save files for document control (source safe) and to send files as an e-mail attachment; for example, you might e-mail an exported file to Technical Support for analysis.

You can export individual metadata objects by right-clicking folders in the EpiCenter Manager directory tree and selecting the Export command from the pop-up menu. This option is available only for exportable metadata objects.

To display a dialog box in which you can export all or any subset of metadata files:

Step 1: Right-click the EpiCenter icon and select Export\Export All from the pop-up menu, which displays the Exporting Metadata dialog box (Figure 72).

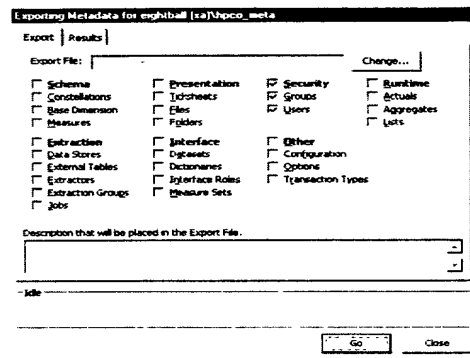


Figure 72: Exporting Metadata Dialog Box

- Step 2:** In the Export tab of the Exporting Metadata dialog box select a heading, such as Schema to select all of the Schema topics automatically, or select individual subtopics.
- Step 3:** Enter any description that will be placed in the export file. This description can be read in the Importing Metadata dialog box when the user selects this file for import.
- Step 4:** Click Go.

To import individual metadata files in EpiCenter Manager, right-click a folder and select Import from the pop-up menu. This option is available only for metadata files that can be imported.

You can produce a new EpiCenter by importing all of the metadata files from an existing EpiCenter:

To import all of the metadata:

- Step 1:** Right-click the EpiCenter icon and select Import Metadata from the pop-up menu.
- Step 2:** The default export file is **export.mdb** in your **ConfFiles** directory. Click Change to select another file.
- Step 3:** Select Always Replace Existing Data if you want the new data to replace existing entries of the same name.
- Step 4:** Select Continue after Errors so that should an error occur, you will receive a partial report.
- Step 5:** Click Go.

Exporting/Importing Metadata

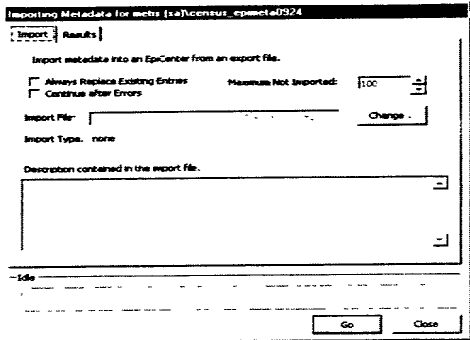


Figure 73: Importing Metadata Dialog Box

09:25:18.072500

Note the following:

- Re-importing objects does not delete references to those objects. For example, a measure definition can be re-imported without deletion of the measure mappings in ticksheets that point to that measure.
- To speed up import and export, the entire Microsoft Access database is either read from or written to at once.
- All import operations are performed inside of a single database transaction. Any error that occurs during import can be rolled back completely.

See Appendix G, “Export/Import of Metadata” for more information.

TOGGLE A AND B TABLES

To switch to the A set of tables from the B set, or vice versa, choose EpiCenter/Toggle A & B Tables, or click the Toggle A & B Tables toolbar button.

WEB BUILDER

As a security measure, the functions necessary for building ticksheets are available through a separate program called Web Builder. Access to only the Web Builder program is appropriate for someone who is authorized to create ticksheets for an already initialized EpiCenter, but does not need access to all of the features of EpiCenter Manager.

During installation this program executable (**WebBuilder.exe**) is placed in the Epiphany directory by default (**C:\Program Files\Epiphany**) and can be selected from the Start menu: Programs\Epiphany*instance_name*\EpiCenter Manager (User Interface Only). After you run the executable, the EpiCenter Enterprise Manager window is displayed. At the top level is the Servers icon. Follow these steps to register your server:

Step 1: Choose Register from the Server menu. The Server Properties dialog box (Figure 9, on page 88) is displayed in the window.

Security Manager

- Step 2:** Select the Server Type from the drop-down list and enter the database server's name, and the username and password for this server.
- Your database server machine icon and the EpiCenters folder (or directory) appear in the hierarchical tree structure. This server icon represents the server where the EpiMart and EpiMeta databases for this EpiCenter reside.
- Step 3:** Select the EpiCenter you plan to work with from the Choose EpiCenter dialog box.
- In the EpiCenter that is displayed, only two folders are shown: the Constellations folder with Measures and Ticksheets only, and the Shared Interface folder, which shows only Glossary Entries and Datasets.
- See "Configuring a Clarity or Relevance Ticksheet" on page 126 for instructions.

SECURITY MANAGER

As a security measure, the security functions of EpiCenter Manager are packaged as a separate program. Security Manager is available for use by those responsible for setting up access rights for users and groups and permissions for ticksheets, but who do not need access to all of the other features of EpiCenter Manager.

During installation this program executable (**SecMgr.exe**) is placed in the Epiphany directory by default (**C:\Program Files\Epiphany**) and can be selected from the Start menu: Programs\Epiphany*instance_name*\EpiCenter Manager (Security Only). After you run the executable, the EpiCenter Enterprise Manager window is displayed. At the top level is the Servers icon. Follow these steps to register your server:

- Step 1:** Choose Register from the Server menu. The Server Properties dialog box (Figure 9, on page 88) is displayed in the window.
- Step 2:** Select the Server Type from the drop-down list and enter the database server's name, and the username and password for this server.

Your database server machine icon and the EpiCenters folder (or directory) appear in the hierarchical tree structure. This server icon represents the server where the EpiMart and EpiMeta databases for this EpiCenter reside.

Step 3: Select the EpiCenter you plan to work with from the Choose EpiCenter dialog box.

The EpiCenter contains only the Security folder. See “Security” on page 200 for instructions.

RUNNING THE SCRUTINY DEBUGGING TOOL

Scrutiny is an interactive debugging tool within EpiCenter Manager. It performs a subset of checks available in the Epiphany Application Server without your having to examine log files, and attempts to fix any problems it finds.

Scrutiny ensures that your EpiCenter (both EpiMeta and EpiMart) is in a consistent and functioning state. It does this by running an extensive set of queries against your EpiMeta and EpiMart to ensure various rules are followed and multiple descriptions of items are consistent. For example, Scrutiny can examine the EpiMart tables for missing indexes, missing rows, bad referential integrity, and so forth. It can also catch Momentum constraints such as not allowing group transaction filters on an individual ticksheet. It will also check consistency constraints, such as the presence of UNKNOWN rows in EpiMart tables and well-formed references.

You can run Scrutiny periodically as a validation tool, or whenever you encounter any problems with metadata or EpiMart data, or if the Application Server fails to start. Scrutiny fixes and diagnoses many common (and less common) issues.

09625518.072500

Running the Scrutiny Debugging Tool

Choose Run Scrutiny from the EpiCenter menu. In the Scrutiny dialog box (Figure 74), select whether you want to perform EpiMeta or EpiMart checks, or both. For EpiMart, you can select the A or B set of tables, or both. In general, Scrutiny runs quickly (a few minutes for large EpiCenters), especially when EpiMart is not selected.

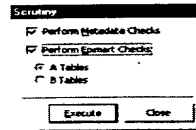


Figure 74: Scrutiny Debugging Tool

When Scrutiny runs, it displays a text screen of the checks that are running. If an error is detected, it describes the nature of the error and either how to fix it yourself (usually in EpiCenter Manager) or proposes a solution for you. If you would like it to execute its proposed solution, press y. After Scrutiny has executed, it asks if you want to re-run that section, to ensure that the fix was applied successfully.

Important: In some cases Scrutiny fixes are last resorts, and should be applied only if all else fails. These checks are identified as such, but be sure to read the descriptions carefully before applying any fix.

EPIPHANY APPLICATION SERVER

The Epiphany Application Server is the component of the Epiphany Application Suite that processes all user requests and returns query data in HTML format. All Epiphany applications run on top of the Application Server. The Application Server has a multi-threaded design that allows several interactive connections to occur simultaneously.

In general, an Application Server interaction begins when the user points a browser at a Web URL address. The URL is first processed by the Web server (IIS), and then routed to the Epiphany proxy, **Epiphany.dll**. The proxy packages the request and sends it to the Application Server via a TCP/IP socket. The proxy waits for the Application Server to process the request and return a result. The proxy then takes the result and returns it to the user's browser via IIS. The proxy serves to separate the Application Server from the IIS process. This architecture allows several IIS machines to point at the same Application Server. The Application Server listens for requests, dispatches them to work or threads, then processes the requests and returns the results. Figure 75 shows the role of the Application Server in the Epiphany system.

An installation may run multiple Application Servers that are installed on one or separate machines. For example, a second Application Server might serve as a backup, or one Application Server might be used for development and another used for production.

Epiphany Application Server

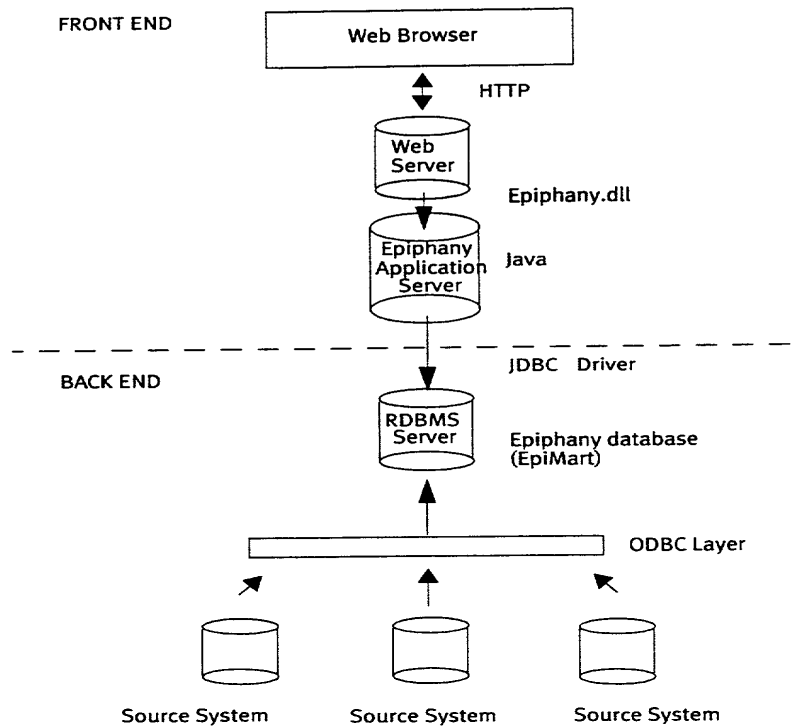


Figure 75: Epiphany System Diagram

The Application Server is a Java application that is normally run as an NT Service. Java is not used on the client (browser) side.

This chapter covers the following topics related to the Epiphany Application Server:

- Starting and stopping (see page 227)
- Command-line arguments (see page 232)
- Refreshing (see page 232)
- EpiAppService program (see page 237)
- Epiphany proxy (see page 243)
- Logging (see page 245)
- Security (see page 250)
- Administrative Groups (see page 256).

For instructions on how to troubleshoot the Application Server, see Appendix H, “Troubleshooting.” This appendix also describes the Application Server error messages.

Note: If the Application Server won’t start, you can run the Scrutiny debugging tool in EpiCenter Manager to diagnose the problem. See “Running the Scrutiny Debugging Tool” on page 223.

STARTING AND STOPPING THE SERVER

This section gives instructions for starting and stopping the Application Server when you run it as a service and when you run it as a console application.

Running as a Service

A standard Epiphany software installation (as described in the *Epiphany Installation Guide*) installs the Epiphany Application Server as an NT service. You can start and stop the service using the Windows *Control Panel\Services* menu. Follow these steps:

Step 1: From the Start menu, choose *Settings\Control Panel\Services*.

Running as a Console Application

Running as a Console Application

Usually, you will start Epiphany Application Server from the Start menu: *Settings\Control Panel\Services* menu. During debugging and the initial setup, however, it may be expedient to start the Application Server from a console window because all of the logging information is copied to the console.

You can manually start the Application Server by using the **jre** program (in your local path) from the directory:

C:\Program Files\Epiphany\instance_name\classes.

Enter the following command line:

```
jre -mx64M -classpath .;.\connect;.\connect.jar;.\
\EpiAppServer.jar;C:\PROGRA-1\
JavaSoft\JRE\1.1\lib\rt.jar com.epiphany.server.Server
localhost:8081 @instancename
```

Notes:

If the **jre** program directory is not in your local path, you will have to pre-pend the pathname to the **jre** command. Normally, **jre** is installed in the **C:\Program Files\JavaSoft\jre\1.1\bin** directory. This command example also assumes that the Application Server is running on the local machine on port 8081 (localhost:8081). The **-mx64M** parameter allows the Application Server to allocate up to 64 megabytes of memory.

For security to function properly, the user who runs this command line must have certain NT user rights. Otherwise, no one will be able to log in (you will receive `EpiLoginException` violations).

When you manually start the Application Server using the **jre** command, a console window that echoes all of the logging information is displayed on your screen. The window must remain open while the Application Server runs. If you close this console window, the Application Server terminates. If you log out of your machine, the console window closes, and the Application Server terminates.

Determining if the Application Server Is Running

```
09625540.075500
--- EPIPHANY APPLICATION SERVER ---
--- Version 3.4.0.1285
--Tue Jan 20 11:05:57 PDT 1999
Instance: momentum_proust3
Adding session: __ global __|unspoofable
[EpiCenter] Initializing instance momentum_proust3
  EpiMeta : lightfoot/hxco33_epimeta0924 {user=sa}
  EpiMart : lightfoot/hxco33_epimart {user=sa}
  Momentum: =====Starting momentum special dimensions load
  Momentum: Got a household dim base of ICN
  Momentum: Got a individual dim base of Site
  Momentum: Finished Loading int map table list
  Momentum: =====Finished momentum special dimensions load
  [TimeNav] Initializing instance momentum_proust3
  [TimeNav] Initialized instance momentum_proust3 in 4426 ms
  Momentum: =====Starting momentum metadata load
  Momentum: Got a measure label of Dollar Amount
  Momentum: Got a measure label of Average Sale Price Shipment
  Momentum: Scanning non-momentum ticksheets for 11 ticksheets.
  Momentum: =====Finished momentum metadata load
[EpiCenter] Closing connections.
[EpiCenter] Initialized instance momentum_proust3 in 16373 ms
AggNav initializing instance momentum_proust3
AggNav initialized instance momentum_proust3 in 1061 ms
Security manager initializing instance momentum_proust3
Security manager initialized instance momentum_proust3 in 630 ms
Storage manager initializing instance momentum_proust3
Storage manager initialized instance momentum_proust3 in 341 ms
[om] Beginning init in C:\PROGRAM FILES\EPIPHANY\MOMENTUM_PROUST\WEB\TEMPLATES\
[om] Processed 86 templates.
Initializing encryption engine...
Server instance created!
Server Host: localhost
Server Port: 8088
Server Inst: momentum_proust3
**** Epiphany AppServer momentum_proust3 awaiting connections... ****
```

Figure 76: Sample Application Server Log File

COMMAND-LINE ARGUMENTS

The Application Server's main routine is the Java program **jre**, which is located in the **com.epiphany.server.Server** class. The **jre** program has the following command-line arguments:

```
jre -mx64M -classpath classpath com.epiphany.server.Server  
    machinename:port number @instancename DEBUG=1
```

If the machine name and port number are not specified, then they are read from the Registry under the *instance_name* directory. See "The Application Server's Registry Keys" on page 240 for more information. If the *instance_name* is not specified, then the server will read the following file to determine the default *instance_name* to use:

HKEY_LOCAL_MACHINE\Software\Epiphany\Instances

The **DEBUG=1** parameter will cause the Application Server to log extra information. This extra information includes the data about the "clean-up" routines that handle timed-out sessions and old command threads that have finished.

REFRESHING THE APPLICATION SERVER

There are two ways to refresh the Application Server: using the **refresh.exe** command line version or the graphical user interface (GUI), **RefreshApp.exe**.

Upon startup, the Application Server reads metadata and Registry information and caches it for use during normal query processing. The metadata that is read includes:

- The Windows Registry
- The *config_master* table in the EpiMeta database
- Aggregate navigation information.

For Authentication to Work

For authentication to work, run the Application Server as a service under the Local System account, or under an administrator account that has special NT privileges. You will receive Error 1314 if privileges are not set. The special NT privileges are—

- Act as part of OS
- Increase quotas
- Replace a process level token

Note: You can use the pass through security module without setting these privileges. See “Authentication Modules” on page 252.

The Local System account already has these privileges, and almost always, installation will be set up to run the Application Server under the Local System account.

To run the Application Server from the command line, follow these steps:

Note: This procedure affects only the local machine, not the NT network.

Step 1: Log in as a user who has local machine administration access.

Step 2: Assign special NT privileges to a user account. To do so, start User Manager and enter the local machine name in the Select Domain dialog box.

Step 3: Choose Policies\User Rights from the menu.

Step 4: Click Show Advanced User Rights and assign the account of the currently logged-in user to have the privileges specified above.

Step 5: Reboot your machine for the privileges to take effect.

Refreshing the Application Server

- EpiCenter Manager or Security Manager was used to change security permissions for a user or group.

Note: The fact that a template file has changed does *not* require a restart or refresh of the server.

If you have determined that you need to refresh the Application Server, there are several ways to proceed. The simplest way to refresh the server is to stop and then restart the Application Server via the Services Control Panel. However, this requires manual intervention, and so would not be suitable for programmatic refresh (after an extraction, for example). This method also interrupts anyone currently using the system.

Another way to refresh the server is by using the **refresh.exe** program that was installed in your **installdir\win32** directory. This program sends a message to the Application Server instructing it to re-read all of the necessary information mentioned above. The Application Server re-parses all of the template files (in case the TemplateDir Registry key was changed) and re-initializes the Security objects.

Refreshing will not interrupt the requests that are currently running on the Application Server. After the refresh has been completed, however, users who were previously logged on will need to log in again. This is because the security restrictions that affect all users' rights have been changed.

You may invoke the **refresh** program via a DOS command-line or open the RefreshApp dialog box and enter command options. Instructions for both methods are given below.

Aggregate navigation is the process by which the Epiphany query machinery determines the optimal aggregate table to use to satisfy an application's request for information. (The *query machinery* is the component of the Epiphany Application Server that communicates with EpiMart and actually issues SQL statements against the DBMS.)

- Time navigation information.
Time navigation is similar to aggregate navigation information. When issuing the backlog queries, the query engine uses time navigator to find the most optimal time path to calculate the backlog.
- The templates used to render Clarity, Relevance, and Momentum result (report/chart/list) pages
- Security Information
- Ticksheet information

When any of the following events occur, you need to restart or refresh the Application Server:

- After an extraction, either the *current_datamart* has changed or there has been a change in a table that is used in a dynamic listbox filter.
- If you Restart the SQL Service *MSSQL Server* on the server on which the Epiphany database (EpiMart) resides while your Application Server was running against that database, then you must restart the Application Server.
- EpiCenter Manager was used to alter the EpiMart.
- Aggregates have been built or rebuilt since the last time the Application Server was started.
- EpiCenter Manager or Web Builder was used to add or modify a ticksheet, measure, or dataset.
- The Windows Registry entries under the following entry have changed since the last time the Application Server was started:

HKEY_LOCAL_MACHINE/Software/Epiphany/*instance_name*

Invoking RefreshApp

Otherwise, **refresh** indicates that it has failed by outputting the response that it does receive, or by indicating that it has failed. For example, here is the output of a negative interaction in which the user/password failed.

```
Refresh [Build 3.4.0.1035]
-----
Connecting to localhost:8081
Sent the REFRESH instruction to localhost
The refresh FAILED because the username/password combination was
invalid.
REFRESH operation took 3365 ms.
```

(The **refresh** program always displays the amount of time that it has taken.)

Invoking RefreshApp

RefreshApp is the GUI version of the **refresh** executable. You may keep the RefreshApp window open and on your desktop during debugging.

Follow these steps to use RefreshApp:

- Step 1:** Choose the Refresh application from the Start menu:
Programs\Epiphany\RefreshApp.
- Step 2:** For the AppServer name, enter the name of the machine on which the Application Server is running.
- Step 3:** For the AppServer Port, enter the port number on which the Application Server is listening.
- Step 4:** For the username and password, specify a valid NT account that has access to the Epiphany Application Suite.
These are the same values required for an end user to log into the top-level Epiphany Web page. In general, any account that is defined in the Security folder of the EpiCenter Manager application will work. See "Security" on page 200 for more information.

The Refresh Command Line

The **refresh** command-line syntax is:

```
refresh localhost port username password
```

where:

localhost	Specifies the name of the machine on which the Application Server is running.
port	Specifies the port number on which the Application Server is listening.
Username and password	Specify a valid administrative account. These are the same values required for an end user to log into the top-level Epiphany Web page. In general, any account that is defined in the Security folder of the EpiCenter Manager application will work. See "Security" on page 200 for more information.

After the **refresh** application has established a connection to the Application Server and sent the appropriate messages, it displays the following message and waits for a response:

Sent the REFRESH instruction to localhost

Normally, the Application Server takes about 30 seconds to refresh before returning an acknowledgment. (Times may vary based on the size of your EpiMeta database and the speed of your network, and other considerations.) Upon receiving this acknowledgment, the **refresh** program displays:

```
Refresh [Build 3.4.0.1035]
-----
Connecting to localhost:8081
Sent the REFRESH instruction to localhost
The refresh SUCCEEDED.
REFRESH operation took 18827 ms.
```

Step 5: Click Refresh to start the application.

Messages display in the bottom of the dialog box that tell you whether the Refresh succeeded or failed (and the time that this event occurred).

Step 6: In case of failure, click View Log to determine what caused the failure. The information that is normally printed to the screen during the execution of **refresh.exe** is displayed in this window.

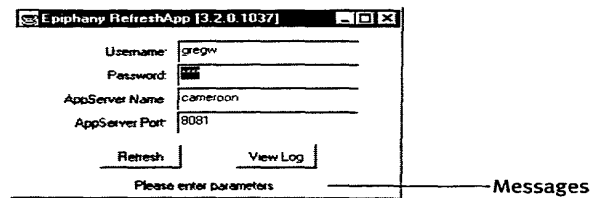


Figure 77: Epiphany RefreshApp Dialog Box

THE EPIAPPSERVICE PROGRAM

The EpiAppService program (**epiappservice**) is the program that Windows NT uses to run the Application Server as an NT Service. You will use EpiAppService to—

- delete an Epiphany Application Server service.
- change the parameters of an Epiphany Application Server service.
- add a new Epiphany Application Server service; for example, create a service because the installation failed, or to install another instance.

Command Syntax

Command Syntax

The **epiappservice** command syntax is—

epiappservice -s servicename action

where *action* is one of the following parameters:

-C Creates a service. Registers a new service with the NT system. You must use the **-E** option with this option to specify the executable to use as the NT Service; for example:

EpiAppService -S "instance" -C -E "*program name*"

When you are configuring an Epiphany Application Server, the command line will look like:

EpiAppService -S "instance" -C -E "C:\Program Files\Epiphany\instance\win32\EpiAppService.exe -S instance"

If the instance name has spaces in it, then you should use single quotes inside of the outside quotes. Do not use spaces within the instance name.

Run the EpiAppService program to create the service, which is also named EpiAppService. It is invoked with the **-s** option to specify the instance name. (The EpiAppService program is both the program that should be run as a service, as well as the program used to register the service.)

-E cmd The command to associate with the service.

00625518.072500

Command Syntax

- DEP *service*** Allows you to specify a service on which the Application Server depends. The Service Manager will always load this dependency before loading the Epiphany Application Server. The service name supplied as an argument to this option must exactly match the service name used by the service in question, usually the Microsoft SQL Server.
- D** Deletes the service specified via the **-s** option from the NT Service Registry. When this service is deleted, only the entry in the NT Registry for the service is deleted; no executables are deleted. Use this to remove unused Application Server instances.
- G** The launch handler for the service. (Go.)
- T** Starts a service. The installation program uses this option to start the Web server (which must be shut down during installation). It starts the service specified via the **-s** option, which is equivalent to clicking the Start button in the Control Panel\Services dialog box to start the desired service. You may use this option when you cannot access the Control Panels (for example, during RCMD or pcAnywhere remote access).
- P** Stops the service specified via the **-s** option. (See the **-T** option.) The installation program uses this option to stop the Web server during the installation.
- K** Checks if the service specified via the **-s** option is running. The program returns a zero return code if the service is running, and a non-zero return code if it is not.
- ?** Displays online help, a description of the command options.

EpiAppService Command Examples

EpiAppService Command Examples

Examples of the most common invocation of **eppiappservice** follow:

- Creation of a service because the installation failed, or you wish to install another instance. If the database server is on a different machine from the Application Server:

```
EpiAppService -S instname -C -E "C:\Program Files\  
Epiphany\instname\Win32\EpiAppService -S instname"
```

If the database server is on the same machine as the Application Server:

```
EpiAppService -S instname -C -E "C:\Program Files\  
Epiphany\instname\Win32\EpiAppService -S instname" -DEP  
MSSQLServer
```

- Deletion of an old service.

```
EpiAppService -S instname -D
```

The Application Server's Registry Keys

All Registry keys used by the Epiphany Application Server are located in **HKEY_LOCAL_MACHINE/Software/Epiphany/Instances/instance_name** where *instance_name* specifies the name of the instance you entered during the Epiphany software installation. This directory contains the following keys:

AppServerHost	The name of the machine on which the Application Server is running. The default is the <i>localhost</i> .
ApplicationServerPort	The port number on which the Application Server is listening for connections. The default is 8081.

AppServerLogVerbosity

This parameter specifies the degree of logging (verbosity level) that the Application Server performs when logging to the database. If the value is 0, no logging is performed. If the value is 1; logging is performed.

AppServerQueryTimeout

The number of seconds before the Application Server automatically terminates long-running queries. A query that requires more than this number of seconds to process will be terminated in order to prevent the exhaustion of system resources and run-away queries from bringing down the server.

AppSessionTimeout

The value in seconds for the lifetime of an idle session. If a session has been idle for this many seconds, then it is removed from the cache.

Build

The version number of the Application Server. This value should be 3.4.

ChartsOutputDir

The directory (full path) in which the *.epc chart files will be stored.

DatabaseName

The name of the EpiMeta database.

DatabaseServer

The name of the database server on which the EpiMeta database is located.

DatabaseUsername

A username for logging into both the EpiMeta and EpiMart databases.

DatabasePassword

The password that corresponds to the DatabaseUsername account.

DatabaseType

Specifies the type of the database.

09625518.072500

The Application Server's Registry Keys

Description	A textual description of the instance.
DSN	The name of the DSN that is used to connect to the EpiMeta database.
InstanceRootDir	The root directory into which the Epiphany Application Server has been installed.
ProxyLogFile	(Optional) Enables proxy logging. For example, setting HKEY_LOCAL_MACHINE\SOFTWARE\Epiphany\Instances\capri\ProxyLogFile to C:\proxy.log will create a proxy.log file if it does not already exist, and log information for every proxy submit to the Application Server. If an invalid path is specified in this ProxyLogFile Registry key, no logging will be performed.
SystemLogDir	The directory in which the Epiphany Application Server log files are written.
SystemLogDirWebpath	The name appended to http://machinename/instance_name/ that informs the Web server of the log files' locations.
TempDirGarbageLifetime	The lifetime in seconds of a temporary or log file created by the Epiphany Application Server. After this specified number of seconds from the creation date, a temporary file, log file, or *.epc chart file will be erased by the Temporary FileManager in the Application Server. Garbage collection occurs at Application Server startup and periodically when there are free cycles.
TemplateDir	The directory in which the Epiphany Application Server templates are stored.

Driver	The JDBC driver used to connect to an EpiMart database. The default is connect.microsoft.MicrosoftDriver . <i>Do not change this value.</i>
SecurityClass	(Optional) The Java class to use for security authentication. If it is not specified, the com.epiphany.security.EpiNTLogon class is used.

THE EPIPHANY PROXY

The Epiphany proxy (**Epiphany.dll**) is an ISAPI application that mediates the requests and responses between the IIS Web server and the Epiphany Application Server. All user requests for Epiphany pages are directed to the proxy **http://machinename/scripts/instance_name/Epiphany.dll**.

This proxy bundles the request into a package that conforms to a strict Epiphany format and sends the package to the Epiphany Application server through a TCP/IP socket. The proxy uses the Windows Registry to find the Epiphany Application Server. In particular, the proxy parses the *instance_name* out of the requesting URL. It then opens the AppServerHost and AppServerPort with that instance's Registry tree, and uses this information to connect to the Application Server. The Application Server processes the request and sends a result back to the proxy. The proxy displays the result in the user's browser.

Note: The *instance_name* in the URL must match the *instance_name* as defined in the Windows Registry (configured from the installation program). This allows one proxy to direct requests to several different Application Server instances. This is of value when you want to have two versions, such as a release and a test instance.

Proxy Logging

The **Epiphany.dll** ISAPI proxy supports rudimentary logging. If you suspect that the proxy is not passing all parameters to the Application Server, you can enable logging of all parameters that the proxy submits.

09625518.072500

Proxy Logging

Proxy logging, which is optional, is a diagnostic tool for identifying problems. not a run-time logging facility. Because the log file may grow arbitrarily large if proxy logging is always enabled, use it only in the event of a suspected problem with the **Epiphany.dll** proxy.

An example of a proxy log file:

```
Processing new request. Header: mGET q p v0.8 oHTTP/1.0 nzhenya
P80 r192.0.0.147 aGodzilla/4.04 [en] (WinNT; I ;Nav) u
U/scripts/capri/Epiphany.dll S/scripts/capri/Epiphany.dll
Requested dispatched. Request data length was 0
Processing new request.
...
Request data length was 72
```

The log file consists of blocks, with each block representing a submit to the proxy DLL. Each new submit starts with `Processing new request. Header:` and is followed by the header. The header consists of header items, each beginning with a letter that describes the type of item, followed by the value of item.

The first letter of the header item is defined below.

m	method POST or GET
q	query string (used with GET)
v	version
o	HTTP protocol
n	server name
P	server port
r	remote IP address
a	user agent

u	user name (if Web server is using authentication)
U	URL
S	script name

The header will be followed by the data section if the submit was of the type POST. First, a number of bytes is printed (172), then the actual data, which should be the same size as the number of bytes printed. Finally, if the submit was successful, the following line is printed:

Requested dispatched. Request data

If the submit failed, then the log file should read:

FAILED TO DISPATCH REQUEST DUE TO A SOCKET ERROR.

APPLICATION SERVER LOGGING

Most components of the Application Server maintain a log file of their activities. This section describes each of these logs, their directory location, and their diagnostic use.

The components that generate logs and the contents of those logs are briefly described below (and in more detail later in this section).

Server

The **com.epiphany.server.Server** class generates a log file of connections made to the Application Server. This file contains the time at which a connection was accepted, the number of that connection, the time at which the request was finished, the total time required by the request, and certain messages printed to the log during the processing of that request. The latter category includes the session ID of the connection, the template used to process that request, the number of bytes generated in the response, exceptions that might be

09625518.072500

Log File Location

generated during the processing of the request, or the user name used to log into the server.

SecurityManager

The **com.epiphany.security.SecurityManager** class generates a log that contains the users and groups in the system. It also captures login and sync-up information.

StorageManager

Logs all function calls, SQL, exceptions, and errors for the storage subsystem.

Clarity, Relevance, or Momentum Query Log

Each Clarity, Relevance, and Momentum request generates a log that contains the submitted ticksheet's parameters, information about how the ticksheet was parsed, all of the SQL statements executed during the processing of the request, and information about how long the processing took.

Log File Location

In the Windows NT Registry on the Application Server machine, the Registry key that specifies the location of all log files generated by the Application Server is named:

HKEY_LOCAL_MACHINE\Software\Epiphany\Instances\instance_name\SystemLogDir

Log File Naming Conventions

All log file names begin with a prefix that indicates the date and time when the log file was first created. The format is as follows:

YYYY-MM-DD_HH-MM-SS-MSname.txt

where YYYY stands for the year, MM for the month, DD for the day, HH for the hour, MM for the minute, SS for the second and MS for the millisecond. *name* can be one of the following: *SRV*, *SECURITY*, *STORAGE*, or *QM_sessionid*. The logs of queries for Epiphany applications have the suffix *QM_sessionid*.

The Server Log

When you start the Application Server, the server object creates a log immediately after the Registry has been opened and read. The Registry must be opened first because it contains the SystemLogDir Registry key that specifies exactly where the log files are stored.

```
**** Epiphany AppServer Acme awaiting connections... ****
Mon Jan 08 14:48:50 PDT 1999:
+++++ Accepted [1] @ [897342530272]
----- Dispatched [1] @ [897342530322]
Free/Total Memory: 165064/3297272
Setting to default template (no app): toplevel
No session exists/created. Displayed login screen.
4865 bytes generated.
----- Finished [1] @ [897342530773]
501ms
```

```
Mon Jan 08 14:48:59 PDT 1999: +++++ Accepted [2] @ [897342539215]
----- Dispatched [2] @ [897342539245]
Free/Total Memory: 719832/3342328
Template passed in: toplevel
Got EP_USER: lslater
Adding session: EPIPHANY\lslater|192.0.0.12
5184 bytes generated.
----- Finished [2] @ [897342539615]
400ms
```


Save and Restore Manager

The next section of this log file contains a list of all the users recognized by the system. Each entry contains a user name, the groups to which the user belongs, and the list of ticketsheets that the user has permission to view. Note that the ticketsheet list is mostly empty since all users belong to at least one group, and therefore inherit ticketsheet permissions.

Save and Restore Manager

The Save and Restore Manager logs all function calls, SQL, exceptions, and errors for the storage subsystem. If you cannot save a file, this is a good place to check.

Epiphany Applications

Note: If the **log** link at the bottom of an HTML Clarity/Relevance/Momentum result page does not take you to a log, make sure that the SystemLogWebDir Registry key refers to an alias that has been configured on the Web server. Also make sure that the directory and files have correct read permissions.

Each Epiphany application creates its own log file before performing any non-trivial processing. The log contains the version of the Application Server and the date/time of the request. The second line begins with **APPLICATION** parameters: and subsequent lines contain all of the name-value pairs submitted to the Application Server for this request. Multiple values submitted with the same name are shown separated by commas. A blank line follows, and then the Application filters, dimensions, and measures that could be parsed from the submitted ticketsheet's parameters are logged.

Next, the system logs all of the SQL statements needed to compute the answer to the request, the amount of time in milliseconds to process the query, and the number of rows returned for all SQL queries.

The log ends with the **CLEANUP** message.

APPLICATION SERVER SECURITY

Authentication, which is performed outside of the Epiphany system, does not capture password information. This external authentication requires an authentication module. For example, the NT authentication module authenticates users with an NT domain controller. The Security Manager inside the Application Server loads the authentication module specified through the SecurityClass Registry key at initialization. Each authentication module supports a fixed API that includes methods to authenticate users, add new users to the Epiphany system, and sync-up outside information on the user (such as group memberships) with Epiphany metadata.

Currently, Epiphany provides two authentication modules: the NT authentication module EpiNTLogon, and an insecure authentication module called EpiPassThruLogon. Use EpiPassThruLogon only for testing and debugging. Optionally, you can add an LDAP authentication module, X.500 module, and other similar modules.

The optional Registry key SecurityClass under the *instance_name* Registry directory controls which security module is loaded. You must specify the full class path to the security module. If this key is omitted, the system uses the default security module **com.epiphany.security.EpiNTLogon**, which uses NT to perform user authentication.

The means by which the authentication information (username and password) reaches the Application Server is as follows. Users log into the Epiphany system either through a login template or through a Web server authentication mechanism, such as Basic Authentication or NTLM.

When the Application Server receives the user name and password, it calls the Security Manager to log in the user. The Security Manager loads an authentication module, and attempts the login process. If the process is successful, depending on the authentication module, one of the following steps is taken:

- If the user exists in the EpiMeta database, user group memberships outside of the Epiphany system will be synched up with group memberships in the EpiMeta database.

- If the user does *not* exist in the EpiMeta database, but is authorized to use the Epiphany system, then a user account will be created in the EpiMeta database. User group memberships outside of the Epiphany system, such as NT, will be synced-up with group memberships in the EpiMeta database. For example, assume your EpiMeta had a group named EPIPHANYUSERS configured with access to all of the ticksheets. If a user name Joe (a valid NT user who belongs to the NT group EPIPHANYUSERS) supplies a correct password, then Joe will be automatically added to the EpiMeta metadata as a user who belongs to the Epiphany group EPIPHANYUSERS.

Only group memberships for a group whose Group Definitions dialog box in EpiCenter Manager has the Synchronize option selected will be synced-up. Sync-up occurs if:

- the user who logs in is 1) a member of a Group X outside of the Epiphany system; 2) Group X exists inside the Epiphany system; and 3) the user is not a member of Group X inside the Epiphany system. Sync-up will create a group membership to Group X for this user inside the Epiphany system.
- the user who logs in is 1) a member of a Group X inside the Epiphany system, but 2) the user is not a member of this group outside of the Epiphany system. Sync-up will remove a group membership from the Epiphany system.

If the sync-up process requires a creation of a new group membership for a user, certain access rights are set on this membership. The ability to save queries has the access rights of Save Group/Default, and global-level save access is Inherit. (See “Security” on page 200 for more information.)

For a group to be the same in EpiMeta and the NT domain, it needs to have the identical name (case sensitive) in both. The group name in the NT domain includes the domain name, such as **Epiphany\Sarah**. You need to make sure that the group name in the EpiMeta includes the domain name for that group.

When the user is authenticated and user information is synced-up, Security Manager determines if the user is authorized to use the Epiphany system. The user is so authorized if he or she belongs to at least one group in the Epiphany system. (The user must also must log in with a valid password.)

09625518 072500

Authentication Modules

Important: An authenticated user is authorized to use the Epiphany system if and only if that user is a member of at least one group in the Epiphany system after sync-up.

Authentication Modules

The following authentication modules are included with the Epiphany software.

Module	Class Name
NT (default)	com.epiphany.security.EpiNTLogon
Pass through	com.epiphany.security.EpiPassThruLogon

- **NT (default)**
NT domain authentication. NT groups are imported into the Epiphany system through EpiCenter Manager. As mentioned, these groups need have the Synchronize option selected in the Group Definition dialog box, which means that memberships to those groups will be synced-up. Users who belong to those groups can log into the Epiphany system. When a user logs in, his or her NT group memberships are synced-up to the EpiMeta database. It is also possible to create Epiphany-only groups inside EpiCenter Manager by simply not selecting the Synchronize option. Thus, the Epiphany administrator may create arbitrarily complex permission hierarchies using EpiCenter Manager, independent of the manner in which NT domain security is set up.

- Pass through (*for in-house debugging purposes only; should never be used at a customer site after the initial Epiphany system setup*)

This is an insecure authentication that has no password. This is an Epiphany-only development module that ignores NT authentication all together. You do not need passwords in this model: specify your user name exactly as it appears in EpiCenter Manager (it must include a domain name if such exists), and you will be logged in. There is no sync-up process. That means that an authenticated user that does not exist in the EpiMeta database will not be allowed to use the Epiphany system. Epiphany groups and users are created and managed through EpiCenter Manager.

Authentication Module Tips

- If you want to use NT-related authentication modules, make sure that **EpiNTLoginJNI.dll** is in your system path.
- Users that do not belong to any groups in the Epiphany system are not allowed to log in. An error message that says that user is authenticated but not authorized to use Epiphany system is displayed whenever an unauthorized but NT-authenticated user logs into the Application Server. User memberships may be adjusted upon login if the user is a member of synchronized groups in Epiphany system. A user must be a member of at least one Epiphany group after the synchronization process completes.
- The optional Registry key SecurityClass (located in the *instance_name* Registry directory) controls which security authentication module is loaded. The full class path to the security module must be specified as the value for this key. If this key is omitted, the default security module is **com.epiphany.security.EpiNTLogon**, which uses NT to perform user authentication.
- The **Toplevel.template** is the template that appears immediately after a user logs into the Epiphany system. The name of this template is configurable with the an optional ToplevelTemplate Registry entry in the **Instances** directory. You can load the **company.template** instead of **toplevel.template** by substituting your company name in ToplevelTemplate=*company*.

Authentication Modules

- Depending on how IIS security is set up, one of the following situations occurs upon login:
 - If Allow Anonymous authentication is enabled, the login dialog box is displayed when the user attempts to use the system for the first time, or the user session times out. It is almost always sufficient to specify the username only. The domain name is located automatically.
 - The search order that authentication uses to find the user account is as follows. First, the authentication mechanism looks in the local machine's Security Access Manager (SAM), then it checks with the primary domain controller, and afterwards checks with trusted domains. If there are multiple users with the same name between domains and/or a local machine that runs Application Server, specify the full user *name-domainname\username*, or the *local_machine_name\username* if the user logs in from a local machine's Security Access Manager (SAM).
 - If Basic Authentication is enabled, the browser displays a login dialog box when the user attempts to access the Epiphany system for the first time. However, when the user's session times out, no re-login is required. The user is automatically logged in again, and a new user session created.
 - If NTLM authentication is enabled, Internet Explorer automatically performs authentication of the user without displaying a login dialog box, although Netscape displays it. If Basic Authentication or NTLM is on, the login dialog box does not appear in the Web browser.

Important: Do not create your own domains. Doing so introduces multiple domains, with the Epiphany machine in one domain and the user accounts of the Epiphany system in another domain. In order for the authentication module **com.epiphany.security.EpiNTLogon** to work properly, a two-way domain trust between the Epiphany domain and the customer domain is required.

If you set up a new domain for the machine that runs the Epiphany Application Server, set up a two-way trust and name the machine and the domain differently. In general, do not use the same string for domain names, machine names, and user names.

- The following applies to Synchronized groups:
EpiNTLogon requires the NT domain for lists of global and local groups. The names of these groups will be matched to the names of the groups in the EpiMeta. A group name will be matched if its domain name and group name match. Therefore, group *foo* will not match to group *EPIPHANY\foo*. (The match is case insensitive for both group name and domain name.) If a user is a member of an NT group *EPIPHANY\foo* and EpiMeta has a group called *epiphany\FOO*, there will be a match.
 - If a user is a member of a local group and the group has a global group as a member, the global group will not be picked for synchronization process. Only the groups for which the user is an immediate member are considered for synchronization process.
 - If a user logs in with an account that is local to the Application Server machine, then a membership to a special group called None is automatically retrieved from the machine's SAM. (Every user in the local SAM has a membership in a special global group called None although this group does not exist on the machine.) For this reason, do not create synchronizable groups called None in EpiCenter Manager.
 - Avoid having the same name for domain names, machine names, and user names. For example, if the Application Server runs on the machine *foo*, and the user called *foo* attempts to log in, access may be denied. If the Application Server is running on a machine named *foo*, and a user named *foo* logs in from the primary domain, or from the local Security Access Manager (SAM) of the machine *foo*, authentication will succeed. If user *foo* logs in from a trust domain, however, the authentication will fail. The only way to log in as *foo* from another domain is to give the full name for the user account upon login: *domainname\foo*.

Administrator Groups

ADMINISTRATOR GROUPS

Epicenter Manager users can make any group an administrator group, and there can be multiple administrator groups in the system. If the user belongs to such a group, that user has special powers when it comes to report/folder and ticksheet access.

Administrator users can save, overwrite, create new, change properties and permissions on any non-special and non-hidden file or folder.

Note: Special folders are top-level folders, such as Public and All Users, or user/group folders or default folders. Hidden folders are the MailTo folder; hidden files are clipboard files.

Administrator users have access to all ticksheets in the system. In the future administrator users will be able to perform more powerful operations than regular users.

09625518.072500

APPENDIX A

EPIPHANY MACROS

This appendix describes the Epiphany-supplied system calls and SQL macros.

Note to sites that use their own database types and SQL macros:

The various fact semantic types (such as Transactional/Statelike) use SUM() operators over the fact columns. New tables are created via SELECT INTO from the result. With decimal data types such as NUMBER(16,4) the precision of the resulting column tends to grow under SQL Server until the column takes up many bytes in the fact table. For this reason, both FACTQTY and FACTMONEY map to the MONEY database type in SQL Server.

SYSTEM CALL MACROS

For the most part, you will use Epiphany system call macros for the transfer of information related to the locations of files and database logins from the EpiMeta database to system calls. Many systems calls such as **mv** and **mkdir** are unable to read a database, and few system calls will be able to read Epiphany-created structures. Rather than leaving each system call to its own means to determine its appropriate information, EpiChannel may pass this information to the system call on the command line.

09625518.072500

System Call Macro Syntax

System Call Macro Syntax

The syntax for system call macros can be either

\$\$VERB[*arg1*, *arg2*, ...]

or

\$\$VERB

Unless stated otherwise, the arguments are the names of roles defined with the job. You may use multiple arguments in most cases, which results in an expansion of each argument. Note that the expansion of both of the following is the same:

\$\$verb[*arg1*, *arg2*]

\$\$verb[*arg1*] \$\$verb[*arg2*]

If the verb does not match one of the special words, no expansion of that verb occurs; for example:

echo \$\$notAverb

expands to

echo \$\$notAverb.

If the argument is a role name, and the job does not define that role, the system call is considered to have an error. The following table describes the Epiphany system call macros.

Note: The Usage column in the following table stands for expected frequency of use.

Table 3: Epiphany System Call Macros

Macro	Purpose	Usage	Description
AGG	Child Procs	High	The name of the Aggbuilder executable in \$\$EPIBIN. For example: \$\$AGG \$\$EXC_ARGS -j \$\$JOB_NAME

Table 3: Epiphany System Call Macros

AGGVERIFY	Child Procs	Low	The name of the aggverify executable in \$\$EPIBIN.
APPSERVERHOST	Registry	Low	The value of this Registry variable.
APPSERVERPORT	Registry	Low	The value of this Registry variable.
CHARTSLOGFILE	Registry	Low	The value of this Registry variable.
CHARTSOUTPUTDIR	Registry	Low	The value of this Registry variable.
DATABASE	Database Login	High	<p>Translates to the name of the database or instance. For example:</p> <pre>isql /S \$\$SERVER[Tests] /U \$\$USER[Tests] /P \$\$PASSWORD[Tests] /d \$\$DATABASE[Tests] /w 300 /i /Q "gen_tests_run"</pre>
DBVENDOR	Database Login	Medium	Translates to the vendor of the database technology.
DEBUG_LEVEL	Command Line	Low	Translates to the current verbosity level of EpiChannel. Use this to pass EpiChannel's verbosity onto the subprocesses it spawns via system calls.
DIRNAME	File ID	Medium	<p>Translates to the directory name of the data store, without the last filename component and without a trailing slash. If the role is working dir, the directory name's last component is a unique subdirectory generated for this particular run of EpiChannel. For example:</p> <pre>echo DIRNAME is \$\$DIRNAME[Working Directory]</pre> <p>(but with no arguments it is \$\$DIRNAME).</p>

System Call Macro Syntax

Table 3: Epiphany System Call Macros

DSN	Database Login	Medium	Translates to the ODBC connection string for the database. This string may be generated even for databases accessed using native API's.
EPIBIN	Child Procs	Medium	The name of the win32 directory under the InstanceRootDir Registry variable.
EXC	Child Procs	High	The name of the extract executable in \$\$EPIBIN.
EXC_ARGS	Child Procs	High	The recognized portions of the extract command line.
EXC_CMD	Child Procs	Medium	The extract program and its arguments other than job name. You can use this to fire sub-extract runs. For example: \$\$EXC_CMD -j performance
EXCVERIFY	Child Procs	Low	The name of the excverify executable in \$\$EPIBIN.
FILENAME	File ID	Medium	Translates to the filename of the data store. If the role is Working Dir, the file name is the name of the EpiChannel log file. For example: echo FILENAME is \$\$FILENAME[Working Directory] (but with no arguments it is \$\$FILENAME).
INSTANCE_NAME	Registry	Medium	The name of the instance's Registry subtree.
INSTANCEROOTDIR	Child Procs	Low	Value of the InstanceRootDir Registry variable.
ISS	Database Login	Low	Translates to a number associated with this source of information.

Table 3: Epiphany System Call Macros

JOB_NAME	Child Procs	High	The name of the current job.
PASSWORD	Database Login	High	Translates to the password associated with the database login.
PATH	File ID	Medium	Translates to the directory name and file name placed together (in the DOS file path format).
PROGRAM_NAME	Child Procs	Low	The name of the current extract program.
REGISTRY_EPIPATH	Registry	Low	Name of the Epiphany Registry key.
REGISTRY_ROOT	Command Line	Low	Translates to the Registry key used by EpiChannel.
SERVER	Database Login	High	Translates to the database host server name (the machine's name) in the case of Microsoft SQL Server, or the SQLNet ID (sid) in the case of Oracle. SERVER and SQLNET are identical in behavior and can be interchanged.
SQLNET	Database Login	High	Translates to the database host server name (the machine's name) in the case of Microsoft SQL Server, or the SQLNet ID (sid) in the case of Oracle. SERVER and SQLNET are identical in behavior and can be interchanged.
USER	Database Login	High	Translates to the username associated with the database login.
VERSION	Database Login	Low	Translates to a release number or string associated with this database's installation.

EPIPHANY SQL MACROS

Major database vendors have developed proprietary extensions to SQL that customers may choose to use in their SQL code because of their features. Epiphany also provides SQL macros for SQL Epiphany products. These macros, which are available as an option to customers, attempt to be database vendor-independent.

Epiphany supplies SQL macros to resolve these issues:

- Support coding of SQL statements that works with the syntax of multiple database engines.
- Support SQL that adapts to the structure of the star schema as defined via the EpiCenter Manager.
- Support extraction of contiguous but non-overlapping subsets from those tables that have dates or ascending identification fields.
- Control the physical characteristics of an Oracle EpiCenter.

Oracle-specific SQL Macros

Oracle-specific SQL macros control the physical characteristics of an Oracle EpiCenter. Oracle tables are stored in a logical entity called a tablespace. Because of the various size requirements of EpiCenter objects, such as fact tables and indexes, dimension tables and indexes, EpiCenter allows you to configure which tablespace is used for each object type. The Oracle-specific SQL macros are translated into tablespace names.

Table 4: Oracle-specific SQL Macros

Macro	Definition
\$\$DIMTABLESPACE	Used for dimension tables (including aggregates and mini-dimensions).
\$\$DIMINDEX_TABLESPACE	Used for dimension indexes (including those on aggregates and mini-dimensions).

Table 4: Oracle-specific SQL Macros

Macro	Definition
\$\$FACTTABLESPACE	Used for fact tables (including aggregates and clusters, as well as temporary objects used during semantics).
\$\$FACTINDEX_TABLESPACE	Used for fact indexes (including those on aggregates and clusters).
\$\$TEMP_TABLESPACE	Used for all Application Server temporary tables (those needed for query post-processing at runtime).
\$\$METATABLESPACE	Expands to the name of the tablespace to use for metadata tables on your system.

Normally, the values for these macros are set to match the names of the tablespaces as they are created by the Oracle initialization script provided by Epiphany as described in the *Epiphany Installation Guide*.

To use alternate names for these tablespaces, you need to run SQL statements such as the following against your EpiMeta database.

```
update translation_Actual set actual_string =  
'your tablespace name' where store_type = 'Oracle' and  
translation_string = 'FACTTABLESPACE'
```

Vendor-independent Macros

Although it is possible to avoid proprietary extensions to SQL, the features they offer may be necessary. As a result, consumers of SQL inherit problems raised by these differences unless they decide to bind themselves to a single database vendor. For example, a function called NVL in one database is equivalent to a function called ISNULL in another database.

03625518.072500

09625518.072500

SQL Macro Usage

Epiphany supports multiple database vendors, and Epiphany-executed SQL statements are a major consumer of SQL features. The Epiphany vendor-independent macros serve to provide some degree of isolation from database vendor differences. Epiphany uses these SQL macros in its own SQL so that it does not need to support different “codelines” of SQL.

Although these macros are available to use by Epiphany customers, their use is optional. Customers concerned only with a single database vendor might avoid all use of these macros. However, customers who use multiple vendors now or plan to in the future may choose to use the Epiphany SQL vendor-independence macros in the SQL statements they create.

SQL Macro Usage

An example of both preferred and less preferable SQL macro usage follows:

Preferred:

```
where COLUMN_FILTER[outfitting_order~,~outfitting_order_key~,~oo]
```

Less preferable:

```
where COLUMN_FILTER[ outfitting_order ~,~ outfitting_order_key  
~,~ oo]
```

The following tables provide a brief explanation of each SQL macro:

- Table 5, “Adaptive SQL Macros,” on page 265
- Table 6, “Extraction Set Identification Macros,” on page 266
- Table 7, “Smart Extraction Macros,” on page 269
- Table 8, “Vendor-independent Macros,” on page 270
- Table 9, “Testing Macros,” on page 284

SQL Macro Notes

The following applies to SQL macro syntax:

- You can determine the “real” translations for most Epiphany macros by issuing the following SQL in an EpiMeta database:
Select * from translation_actual
- For usage examples of most of the SQL macros, see the initialization file **templates.sql**.
- The syntax for Epiphany SQL macros may be either **\$\$Macro[arg1~,~ arg2~,~ ...]** or **\$\$Macro**
- The ~,~ is used to separate the arguments if there are multiple arguments.
- If an argument consists of multiple elements, separate the elements using tilde-comma-tilde. For an example, see the macro **CREATE_INDEX_IF_NOT EXISTS** given in Table 8, “Vendor-independent Macros,” on page 270.
- In some of the macro examples, column spaces have been added near the brackets and near the ~,~ entries for better formatting. In reality, any spaces that are present would be forwarded into the final SQL, so the best coding practice is to avoid them.
- The Usage column in the following tables stands for expected frequency of use.

Table 5: Adaptive SQL Macros

Macro	Usage	Description
CURR	High	Expands the _A or _B suffix of the currently active tables. Allows you to reference the active or new EpiMart tables.
NEXT	High	Expands the _A or _B suffix of the not-currently-active tables.

Table 6: Extraction Set Identification Macros

Macro	Usage	Description
COLUMN_FILTER [table_name ~.~ column_name ~.~ alias_name]	High	Expands to a "one-sided" SQL comparison expression that requires that alias_name.column_name be greater than or equal to the value in table table_name column column_name as of the start of the last run. This extracts "everything since last time." Although the alias name is optional: you should use it.
COLUMN_LAST_VALUE [tbl ~.~ col]	High	Expands to the value of this column as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
COLUMN_RANGE_FILTER [table_name ~.~ column_name ~.~ alias_name]	High	Expands to a "two-sided" SQL comparison expression that requires alias_name.column_name to be greater than or equal to the value in table table_name column column_name as of the start of the last run, and less than or equal to this value as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running."

09625518.072500

Table 6: Extraction Set Identification Macros

DATE_FILTER[column_name]	High	<p>Expands to a "one-sided" SQL comparison expression that requires the column to be greater than or equal to the "current date/time" as of the start of the last run. This extracts "everything since last time."</p> <p>No table or alias arguments are available to the macro. If the column needs to be qualified, just add the qualifications in the first argument. For example:</p> <p>and \$\$DATE_FILTER[oo.date_key]</p>
DATE_RANGE_FILTER [column_name]	High	<p>Expands to a "two-sided" SQL comparison expression that requires the column to be greater than or equal to the "current date/time" as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running."</p> <p>This compares SQL Server datetimes. The column used for the comparison must be declared as a datetime or some variant in SQL Server. Only the date portion of the value is used; the time portion is discarded.</p>
DATE_LAST_VALUE	High	<p>Expands to the "current date" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.</p> <p>This compares SQL Server datetimes. The column used for the comparison must be declared as a datetime or some variant in SQL Server. Only the date portion of the value is used; the time portion is discarded.</p>

Table 6: Extraction Set Identification Macros

TIMESTAMP_FILTER [column_name]	High	<p>Expands to a "one-sided" SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run. This extracts "everything since last time."</p> <p>This compares SQL Server timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type in SQL Server, not as a time or date.</p>
TIMESTAMP_FILTER_RANGE [column_name]	High	<p>Expands to a "two-sided" SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts "everything since last time, but not including that data that changes while the extractions are running."</p> <p>This compares SQL Server timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type in SQL Server, not as a time or date.</p>
TIMESTAMP_LAST_VALUE	High	<p>Expands to the "current timestamp" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.</p>

Table 6: Extraction Set Identification Macros

YYYYMMDD_FILTER [column_name]	High	This is the same as a DATE_FILTER except that only the "day" portion of the date times is used. (Some business semantics are most meaningful when applied to "days.") This extracts "everything since the last day, including the last day."
YYYYMMDD_FILTER_RANGE [column_name]	High	This is the same as a DATE_FILTER_RANGE except that only the "day" portion of the date/times is used. (Some business semantics are most meaningful when applied to "days.") This extracts "everything since the last day, including the last day, but not including today."
YYYYMMDD_LAST_VALUE	High	Expands to the "current date in YYYYMMDD format" as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.

Table 7: Smart Extraction Macros

Macro	Usage	Description
METADBNAME	Medium	Returns the name of the metadata database (EpiMeta).
MARTDBNAME	Medium	Returns the name of the datamart database (EpiMart).
INITIAL_LOAD	Medium	Expands to its argument if the initial load flag is set, or else expands to nothing.
NOT_INITIAL_LOAD	Medium	Expands to its argument if the initial load flag is set, or else expands to nothing.

Table 8: Vendor-independent Macros

Macro	Usage	Description
ADD_DAYS	Medium	Returns a date representation of its first argument plus its second argument as a number of days: For example: SELECT \$\$ADD_DAYS[\$\$DBNOW ~.~ 1]
ADD_MONTHS [date_expression ~.~ number]	Medium	Takes two arguments: adds the second as a number of months to the first argument, which is a date.
ASSERT_INDEX_EXISTS [index_name]	Low	Causes a SQL error if the index named in argument 1 does not exist. For example: \$\$BEGIN_ASSERT_INDEX \$\$ASSERT_INDEX_EXISTS[XPKCustomerMap_B] \$\$ASSERT_INDEX_EXISTS[XPKApplicationMap_B] \$\$END_ASSERT_INDEX
BEGIN_ASSERT_INDEX	Low	In Oracle, declares the DECLARE INDEX_NOT_EXISTS exception. See ASSERT_INDEX_EXISTS.
BOOL_TO_YN[testvalue]	Low	Returns the string N equals 0. Otherwise, returns 'Y'. For example: \$\$BOOL_TO_YN[6] becomes 'Y'.

Table 8: Vendor-independent Macros

CASE_BEGIN	Medium	<p>Begins a case statement that compares an expression to a list of options and returns the value for the first matching option. Also provides a single option value pair.</p> <p>For example:</p> <pre>SELECT \$\$CASE_BEGIN[col_name ~.~ option1 ~.~ val1] \$\$CASE_ELSEIF[col_name ~.~ option2 ~.~ val2] \$\$CASE_ELSE[else_val]\$\$CASE_END</pre>
CASE_ELSE	Medium	<p>Fall-through value for a case statement that compares an expression to a list of options and returns the value for the first matching option. See CASE_BEGIN.</p>
CASE_ELSEIF	Medium	<p>Continues a case statement that compares an expression to a list of options and returns the value for the first matching option. See CASE_BEGIN.</p>
CASE_END	Medium	<p>Ends a case statement that compares an expression to a list of options and returns the value for the first matching option. See CASE_BEGIN.</p>
CAT	High	<p>Used as an operator to append "two" \$\$CAT 'strings'.</p> <p>For example:</p> <pre>\$\$TO_CHAR[table1.col1] \$\$CAT '-' \$\$CAT \$\$TO_CHAR[col2]</pre>

SQL Macro Notes

Table 8: Vendor-independent Macros

CBIN_VAL[testval ~. lowerbound ~. upperbound ~. binletter` CBIN_END	Medium	Can be used to "bin" numeric values into character buckets. Multiple \$\$CBIN_VAL macros should be followed by a single \$\$CBIN_END. If testval is in the range lowerbound to upperbound (inclusive), then the expression yields binletter. For example: \$\$CBIN_VAL[7 ~. 1 ~. 5 ~. 'A'] \$\$CBIN_VAL[7 ~. 6 ~. 10 ~. 'B'] \$\$CBIN_END returns the value 'B'.
COUNTER	Medium	Returns an integer column that counts the returned rows. SELECT \$\$COUNTER the_counter.
CHAR_1	Medium	Expands into a type definition for a single character field.
COUNT_ROWS_FROM COUNT_ROWS_ SELECT	Low	Can be used to count the number of rows in a table. Uses sysindexes on SQL Server for fastest count. For example: SELECT \$\$COUNT_ROWS_SELECT the_count \$\$COUNT_ROWS_FROM[MyTable]
CREATE_INDEX_IF_ NOT_EXISTS [index_type~. index_name~. table_name~. column_list~. after_creation_clause]	Low	Creates an index if it is not already there. \$\$DDL_BEGIN \$\$CREATE_INDEX_IF_NOT_EXISTS UNIQUE ~. XPK_123 ~. table1 ~. ss_key . iss . date_key . transtype_key . seq ~.] \$\$DDL_END

Table 8: Vendor-independent Macros

DBNOW	High	<p>Returns the date/time from the database engine. For example:</p> <pre>SELECT Col1 ss_key \$\$DBNOW date_modified from zork</pre>
DDL_BEGIN	Low	<p>Starts a block of code that changes the schema. Use when there is no DECLARE statement already started. For example:</p> <pre>\$\$DDL_BEGIN \$\$NOT_DEBUG[\$\$DROP_TABLE_IF_EXISTS[table1]] \$\$DDL_END</pre>
DDL_BEGIN_NO_DECLARE	Low	<p>Starts a block of code that changes the schema. Use when there is a DECLARE statement already started. For example:</p> <pre>DECLARE \$\$VAR[transFIXED] \$\$VARCHAR_50\$\$EOS \$\$DDL_BEGIN_NO_DECLARE \$\$VAR_ASSIGN_BEGIN[transFIXED] SELECT \$\$TO_CHAR[transtype_key] \$\$VAR_ASSIGN_INTO[transFIXED] FROM Transtype_O WHERE name = FINV_ADJUST \$\$VAR_ASSIGN_END</pre>

Table 8: Vendor-independent Macros

DDL_END	Low	<p>Ends a block of SQL that changes the schema. For example:</p> <pre> \$DDL_BEGIN \$DROP_TABLE_IF_EXISTS[\$FCTTBL[]\$NEXT] \$DROP_TABLE_IF_EXISTS[\$FCTTBL[]_INC] \$DDL_END </pre>
DDL_EXEC [statement]	Low	<p>All items in the argument list are evaluated at runtime, not when the statement is parsed. This macro can construct SQL based on the values of variables computed in the same SQL block. For example:</p> <pre> \$DDL_BEGIN \$DDL_EXEC[CREATE INDEX X_table1 ON table1 (iss, ss_key, date_key)] \$DDL_END </pre>

Table 8: Vendor-independent Macros

DECLARE_BEGIN	Low	<p>Starts a DECLARE block. For example:</p> <pre> \$\$DECLARE_BEGIN \$\$DECLARE_BODY[\$\$VAR[count_INC] \$\$EPIINT] \$\$DECLARE_BODY[\$\$VAR[count_FC] \$\$EPIINT] BEGIN \$\$VAR_ASSIGN_BEGIN[count_INC] SELECT COUNT(1) \$\$VAR_ASSIGN_INT0[count_INC] FROM \$\$FCTTBL[]_INC \$\$VAR_ASSIGN_END </pre>
DECLARE_BODY [argument]	Low	Treats its argument as a declaration. See DECLARE_BEGIN.
DROP_INDEX[table_name ~.~ index_name]	Medium	<p>Drops the index. For example:</p> <pre> \$\$DDL_BEGIN \$\$DROP_INDEX[table1 ~.~ index_name] \$\$DDL_END </pre>
DROP_TABLE_IF_EXISTS [table_name]	Medium	<p>Drops the table without returning an error indicating that the table does not exist. For example:</p> <pre> \$\$DDL_BEGIN \$\$DROP_TABLE_IF_EXISTS[table1] \$\$DROP_TABLE_IF_EXISTS[table2] \$\$DDL_END </pre>
ELSE	Medium	The start of the negative clause of an IF statement.
END_ASSERT_INDEX	Low	Ends a block of checks that indexes exist. See ASSERT_INDEX.

Table 8: Vendor-independent Macros

END_IF	Medium	Ends an IF statement. see IF.
EOS	Low	Ends a SQL statement. SELECT PROCESSED: COUNT(1), 1100 FROM table1 \$\$EOS
EPIINT	Medium	Declares an int. For example: \$\$DECLARE_BEGIN \$\$DECLARE_BODY[\$\$VAR[unjoined] \$\$EPIINT] \$\$DECLARE_BODY[\$\$VAR[processed] \$\$EPIINT]
EPIKEY	Medium	Declares an Epiphany dimension key. See EPIINT.
FACTMONEY	Medium	Declares a monetary value. See EPIINT.
FACTQTY	Medium	Declares a decimal value. See EPIINT.
FLOAT	Medium	Declares a float value. See EPIINT.
IDENTITY	Medium	Declares a integer serial sequence. See EPIINT.
IF [condition]	Medium	Performs a conditional action. For example: \$\$IF[\$\$VAR[fc_exists] = 0] \$\$DDL_EXEC[\$\$SELECT_INTO_BEGIN[temp_table] SELECT * \$\$SELECT_INTO_BODY[temp_table] FROM Old_table WHERE 1=0] \$\$END_IF \$\$DDL_END

Table 8: Vendor-independent Macros

IJ_FROM [table_name1 ~,~ table_name2]	Low	Performs an inner join on two tables. See JOIN_WHERE .
INSTR [s1 ~,~ s2]	Medium	Returns the position of s1 in s2. For example: \$\$INSTR [b'~,~ 'abc'] returns 2.
JOIN_LEFT_OUTER	Medium	Produces a condition for outer joining the first argument to the second. For example: SELECT ... WHERE \$\$JOIN_LEFT_OUTER [t1.c1 ~,~ t2.c2]
JOIN_RIGHT_OUTER	Medium	Produces an equals sign appropriate for right outer joins (No arguments are needed.) For example: SELECT ... WHERE t1.c1 \$\$JOIN_RIGHT_OUTER t2.c2

Table 8: Vendor-independent Macros

JOIN_WHERE [join_condition]	Low	Supplies the WHERE clause for a join. For example: SELECT col1, col2 FROM Table1 s \$\$LOJ_FROM[table2 m ~,~ s.iss = m.iss AND s.col2=m.col2] \$\$LOJ_FROM[table3 d ~,~ m.col1 = d.col1] WHERE 1=1 \$\$JOIN_WHERE[m.col1=d.col1(+)] \$\$JOIN_WHERE[s.iss = m.iss (+) AND s.col2 = m.col2 (+)]
LENGTH[s]	Medium	Returns the length of a string. For example: \$\$LENGTH['abc'] returns 3.
LOJ_FROM [join_condition]	Low	Performs a left outer join. See JOIN_WHERE.
MAX_SYS_DATE	Medium	Returns the highest date supported by the database.
MODULO{x ~,~y}	Low	Returns the remainder when x is divided by y. For example: MODULO{7 ~,~ 4} returns 3.

Table 8: Vendor-independent Macros

NBIN_VAL[testval ~.~ lowerbound ~.~ upperbound ~.~ binnumber] NBIN_END	Medium	Can be used to "bin" numeric values into numeric buckets. Multiple \$\$NBIN_VAL macros should be followed by a single \$\$NBIN_END. It testval is in the range lowerbound to upperbound (inclusive), then the expression yields binnumber. For example: \$\$NBIN_VAL[7 ~.~ 1 ~.~ 5 ~.~ 1] \$\$NBIN_VAL[7 ~.~ 6 ~.~ 10 ~.~ 2] \$\$NBIN_END return the value 2.
NO_FROM_LIST	Medium	Supplies the "dummy" FROM clause needed by some database vendors. For example: SELECT 'MODIFIED'. \$\$VAR[modified]. 1050 \$\$NO_FROM_LISTS\$EOS
NUMBER(9)	Medium	Declares a decimal(9). See EPIINT.
NUMBER(9.2)	Medium	Declares a decimal(9.2). See EPIINT.
NUMBER(9.5)	Medium	Declares a decimal(9.5). See EPIINT.
NVL [expression ~.~ value]	High	When the first argument is NULL, replace it with the value in the second argument. For example: SELECT \$\$TO_CHAR[\$\$NVL[MAX(col1) ~.~ 1]]
ORACLE [expression]	High	Expands to nothing if the database is not Oracle. For example: SELECT COUNT(1) FROM \$\$\$SQLSERVER[sysobjects]\$\$ORACLE[tabs]

Table 8: Vendor-independent Macros

RAISE_EXCEPTION [Exception]	Low	Raises the given exception (as a variable on Oracle, as a string on SQL Server). For example: \$\$RAISE_EXCEPTION[MyException] raises an exception.
RENAME_OBJECT	Medium	Renamed tables or other database objects. For example: \$\$RENAME_OBJECT[oldtablename ~,~ newtblname]
SELECT_INTO_BEGIN [table_name]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. For example: \$\$SELECT_INTO_BEGIN[temp_tab] SELECT * \$\$SELECT_INTO_BODY[temp_tab] FROM Old_tab WHERE 1=0
SELECT_INTO_BODY [table_name]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. See SELECT_INTO_BEGIN.
SMALLDATE	Medium	Declares a SMALLDATETIME. See EPIINT.
SMALLINT	Medium	Declares a double-byte integer. See EPIINT.
SQLSERVER [expression]	High	Expands into nothing if the database engine is not SQL Server. For example: SELECT COUNT(1) FROM \$\$\$SQLSERVER[sysobjects]\$\$ORACLE[tabs]

Table 8: Vendor-independent Macros

SSKEY	Low	Declares the type Epiphany uses for ss_keys. See EPIINT.
SUBSTRING [expression ~.~ start ~.~ length]	Medium	Performs a substring operation. For example: \$\$\$SUBSTRING[name ~.~1 ~.~8]
SUPERNVL	Medium	Converts a null value for the first argument into the second argument. The resulting column is NOT-NULL able in the schema definition of the result set. For example: SELECT \$\$\$SUPERNVL[col1 ~.~ UNKNOWN]
TABLE_EXISTS_ CONDITION [table_name]	Low	Detects if a table exists. For example: SELECT COUNT(1) FROM \$\$\$SQLSERVER[sysobjects]\$ORACLE[tabs] WHERE \$\$TABLE_EXISTS_CONDITION[table_name]
TABLE_WITH_PREFIX [database_name ~.~ table_name]	Medium	Qualifies a table name with a database or user name. For example: SELECT source_system_key iss FROM \$\$TABLE_WITH_PREFIX[\$\$METADBNAME ~.~ source_system]
TINYINT	Medium	Declares a single-byte integer. See EPIINT.
TO_CHAR [expression]	High	Converts a value to a character. Length is second argument. For example: SELECT \$\$TO_CHAR[\$\$NVL[MAX(col1) ~.~ 1]]

Table 8: Vendor-independent Macros

TO_DATE [expression]	Medium	Converts a value to a database date.
TO_DATEFMT [expr ~.~ format]	Low	Converts expression to a date type with the appropriate Oracle format (<i>format</i> is ignored on SQL Server).
TO_EPIDATE [expression]	High	Converts a date to the string format preferred by EpiChannel. For example: Select coll ss_key. \$\$TO_EPIDATE[date_col] date_modified from zork
TO_NUMBER	Medium	Converts an expression to a number. For example: \$\$TO_NUMBER['123'] returns 123.
TO_TIME	Medium	Converts its argument to a time representation. For example: SELECT \$\$TO_TIME[\$\$DBNOW]
TO_YYYYMMDD [expression]	Medium	Converts a data to a YYYYMMDD string.

Table 8: Vendor-independent Macros

TRANSLATE_BEGIN TRANSLATE_VAL [expression ~.~ searchval ~.~ translationval ~.~ OtherTerm TRANSLATE_ELSE TRANSLATE_END	Medium	Searches expression for occurrences of each searchval and returns the translationval. Note that \$\$TRANSLATE_VAL terms should be nested inside of each other. An optional \$\$TRANSLATE_ELSE can be nested inside the final \$\$TRANSLATE_VAL. For example: \$\$TRANSLATE_BEGIN \$\$TRANSLATE_VAL['abcdef' ~.~ bce ~.~ 'Value1' ~.~ \$\$TRANSLATE_VAL['abcdef' ~.~ cde ~.~ 'Value2' ~.~ \$\$TRANSLATE_ELSE['Other']]] \$\$TRANSLATE_END returns 'Value2'
VAR [variable_name]	Medium	References a database variable. For example: SELECT PROCESSED: \$\$VAR[processed]. 1100 \$\$NO_FROM_LISTS\$EOS
VAR_ASSIGN_BEGIN [variable_name]	Medium	Assigns to a database variable. For example: \$\$VAR_ASSIGN_BEGIN[max_key] SELECT \$\$TO_CHAR[\$\$NVL[MAX(coll) ~.~ 1]] \$\$VAR_ASSIGN_INT0[max_key] FROM table2 \$\$VAR_ASSIGN_END.
VAR_ASSIGN_END	Medium	Assigns to a database variable. See VAR_ASSIGN_BEGIN.
VAR_ASSIGN_INT0 [variable_name]	Medium	Assigns to a database variable. See VAR_ASSIGN_BEGIN.
VARCHAR_100	Medium	Declares a variable-width character datatype that holds a maximum of 100 characters. See EPIINT.

Table 8: Vendor-independent Macros

VARCHAR_15	Medium	Declares a variable-width character datatype that holds a maximum of 15 characters. See EPIINT.
VARCHAR_25	Medium	Declares a variable-width character datatype that holds a maximum of 25 characters. See EPIINT.
VARCHAR_255	Medium	Declares a variable-width character datatype that holds a maximum of 255 characters. See EPIINT.
VARCHAR_5	Medium	Declares a variable-width character datatype that holds a maximum of 5 characters. See EPIINT.
VARCHAR_50	Medium	Declares a variable-width character datatype that holds a maximum of 50 characters. See EPIINT.

Table 9: Testing Macros

Macro	Usage	Description
DEBUG	Low	<p>If debugging is enabled: Expands to nothing if the extract command's verbosity level is less than 3; otherwise, returns its argument.</p> <p>Note: For testing that depends on whether debugging is on or off, use both DEBUG and NOT_DEBUG.</p>
NOT_DEBUG	Low	<p>If debugging is NOT enabled: Expands to nothing if the extract command's verbosity level is higher than 3; otherwise, returns its argument.</p> <p>Note: For testing that depends on whether debugging is on or off, use both DEBUG and NOT_DEBUG.</p>

APPENDIX B

EPICENTER CONFIGURATION

The Configuration dialog box shows the various configurations for your EpiCenter.

To view or modify configuration settings, choose Configuration from the EpiCenter menu. The Configuration dialog box (see Figure 78) is displayed. It has five tabs: General, Transaction Types, Measure Units, Option Labels, and Ticksheet Types.

The Configuration dialog box is the user interface for the *config_master* metadata table that contains various system parameters in name/value format.

GENERAL SETTINGS

The configuration data that appears in the General tab of the Configuration dialog box has been set for a default EpiCenter. Other than entering your e-mail password, the information should be correct, or require minimal alteration.

You may modify these settings if necessary. Select the key in the list and enter a new value in the Value textbox, and click Update. All of the values are defined in the dialog box.

Note the following:

- You need to change the Mail Password after installing the Epiphany software.
- The Mail Profile name is the default e-mail address for the EpiCenter Manager user.

General Settings

- **current_datamart A / B**
Indicates which set of tables (A or B) is active. See “Mirroring: A and B Tables” on page 83.
- **date_type**
Calendar is the default.
date_445 represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.
- **end_year**
The ending year of the EpiMart. The date range of the EpiMart should be at least as large as any dates that are found in the data you will be extracting.
In order for users to get the best results when forecasting trends using Relevance, you need to build the date dimension as far into the future as you plan to forecast. (Currently, the maximum prediction is three years past the last date that has recorded data.) If the date dimension is not built out far enough, the user will receive columns with names such as
Dec 1999, Second Next, Third Next instead of *Dec 1999, Dec 2000, Dec 2001*.
- **fiscal_year start_m[onth]**
If the chosen month is other than January, then the actual starting year is one less than the start_year value (thus the given year may start on January 1 and still be a complete fiscal year).
- **number_of_years**
The number of years during which the data warehouse will be operational. The default is 20.
- **min_sample_invlog10**
The minimum compression ratios for Momentum sampling. If set to zero, then no sampling occurs. If set to 2, then 1/100 of the sampling occurs. The value 3 means 1/1000th, and the value 1 means 1/10th of the sampling occurs, and so forth.

- **mom_inlist_limit**
The maximum number of elements that your system allows in the list used by a SQL IN clause. By default this value is 254 on Oracle and 2,000 on SQL Server.

For example, for a query of the type:

```
SELECT *FROM table WHERE attr#i in list of values
```

mom_inlist_limit specifies the maximum number of values that can be placed in the *list of values* in the above query.

Note: Exporting or importing deletes the value set for mom_inlist_limit. If you export metadata from a SQL Server EpiCenter to an Oracle metadata (or vice versa), you need to re-enter this value.

- **start_day_445**
The beginning day of the 445 quarter.
- **start_year**
The year that the EpiMart becomes operational. The default is 1990.
- **version**
The version number of this EpiCenter Manager.
- **week_start_day**
The day of the week that is the first day of the week. Sunday is the default.
- **last_extract_date**
The date that appears as the *Data is valid as of...* in Clarity reports.

09625518.072500

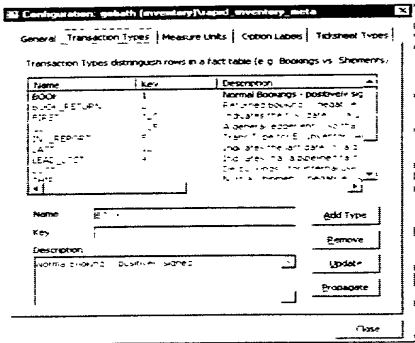


Figure 79: Configuration Dialog Box: Transaction Types

MEASURE UNITS

Measure units define how currency units, percentages, and numerical units are displayed by default on ticksheets. To define a measure unit, enter its name and symbol, such as the dollar or percent sign. The symbol character can be at most one character. To input the Yen sign and other similar characters, use the Windows character map to copy and paste the symbol.

For a currency unit, if this EpiCenter uses currencies from different countries, select the Multi-currency option. Multi-currency controls how the system handles currencies. (A GROUP BY is forced on the currency dimension so that the system does not calculate different currencies, such as French francs and British pounds, as the same currency.) Also, select Postfix if the symbol should follow the number: 100% is postfix, whereas \$100 (the default) is prefix.

09625518.072500

Option Labels

Add any description for your reference in the Description textbox, and click Add Unit. The system generates a hidden key for the unit. EpiCenter Manager uses this key when flagging a measure with a unit designation.

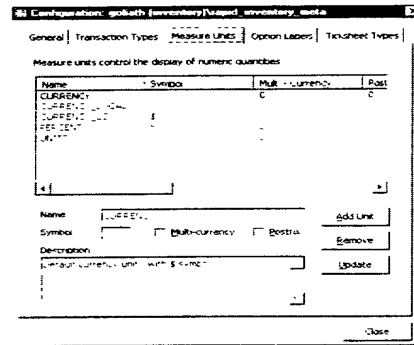


Figure 8o: Configuration Dialog Box: Measure Units Tab

OPTION LABELS

The Option Labels tab lists the labels for the option names as they appear to end users on ticksheets. Although the option names are set by the Epiphany system, you can customize option labels for your ticksheets. For example, you could change the option label for the option name *UnitQuantity* to *Quantities*. To do this, select the Option Name in the upper panel of the tab, enter the name you want to appear on the ticksheet in the Option Label textbox, and click Update.

The Options Labels tab also contains names and values that need to be configured for Momentum ticksheets.

MOMENTUM LABELS

As part of configuring Momentum, you need to modify the default option labels and default value names and labels. Use the upper panel of the Options Labels tab to modify option labels, and the lower panel of the tab to modify value names and labels for the selected option name.

The following option names are specific to Momentum:

- **HOUSEHOLD_CARDINALITY**
Corresponds to the drop-down list that appears on top of the filter pop-up window when one creates a list of households. The value names and default labels are as follows:
 - 0 = For household (select when filtering on household information).
 - 1 = No one in household (select when filtering on individual information—no one).
 - 2 = No more than one in household (select when filtering on individual information—no more than one).
 - 3 = Exactly one in household (select when filtering on individual information—exactly one).
 - 4 = At least one in household (select when filtering on individual information—at least one).
 - 5 = Everyone in household (select when filtering on individual information—everyone).
- **INDIVIDUAL_CARDINALITY**
Corresponds to the label on the top of the filter pop-up window when creating a list of individuals. The value names and default labels are as follows:
 - 0 = No cardinality (you can change this to something such as “Individual filter”).
- **MOMENTUM_HOUSEHOLD_LABELS**
Miscellaneous labels used on the Momentum household filter pop-up window. The value names and default labels are as follows:
 - filter_title: Filter Title (title on top of household filter pop-up window).

- 1 = Some thing (label next to radio button for positive case).

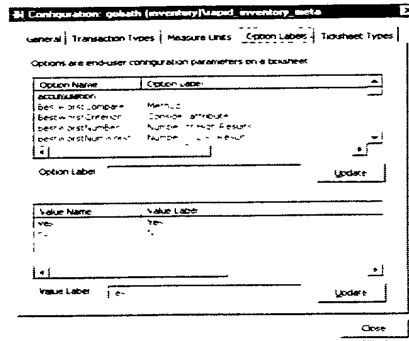


Figure 81: Configuration Dialog Box: Option Labels Tab

Note: When configuring Momentum for an EpiCenter, you can use the Ticksheet Types tab of the Configuration dialog box (Figure 82) to change the ticksheet types labels from their default values, which are Filter Individuals and Filter Households. These are the labels that display in the main menu of the browser window (see Figure 28, on page 129).

TICKSHEET TYPES

The Ticksheet Types tab is primarily read-only and shows the contents of the various ticksheet types. You can update the ticksheet type label, which is the name of the ticksheet types that display in the main menu of the browser window (see Figure 28, on page 129). When working with Momentum and configuring the terminology for Individual and Group, you need to change these labels so that the wording is correct for your installation.

This tab is the means by which you disable an entire application by choosing the application, such as Momentum, and deselecting Enabled. This is useful if you want to test Clarity and Relevance ticksheets, but have not yet set up Momentum.

Ticksheet Types

Configuration - global_inventory/Maped_inventory_module

General | Transaction Types | Measure Units | Option Labels | Ticksheet Types

Each distinct Epiphany module is defined by a ticksheet type

Application: ☒ Enabled

Ticksheet Type	Label	Template	Co
clarity	Table	clarityticksheet	co

Type Label:

Attribute Role
Clarity Column
Clarity Row

#	Option Name
1	Unit Column
2	Unit Label
3	Unit
4	Percent
5	Time Sort
6	Time To
7	Time From
8	Time To

Figure 82: Configuration Dialog Box: Ticksheet Types Tab

09625518.072500

APPENDIX C

DATE DIMENSION FIELDS

The date dimension fields used by the Epiphany system are described below.

Table 10: Date Dimension Fields

dim_col_name	Description
cq_and_cy_name	Calendar quarter and year name. For example, Q1 1999.
cq_name	Calendar quarter name. For example, Q1.
cy_name	Calendar year name. For example, 1999.
date_key	Primary key—date as a native date type.
day_cq_begin	Whether or not this is a day on which a calendar quarter begins. (1/0).
day_cq_end	Whether or not this is a day on which a calendar quarter ends. (1/0).
day_cy_begin	Whether or not this is a day on which a calendar year begins. (1/0).
day_cy_end	Whether or not this is a day on which a calendar year ends. (1/0).

Date Dimension Fields

Table 10: Date Dimension Fields

day_fq_begin	Whether or not this is a day on which a fiscal quarter begins. (1/0).
day_fq_end	Whether or not this is a day on which a fiscal quarter ends. (1/0).
day_fy_begin	Whether or not this is a day on which a fiscal year begins. (1/0).
day_fy_end	Whether or not this is a day on which a fiscal year ends. (1/0).
day_month_begin	Whether or not this is a day on which a month or period begins. (1/0).
day_month_end	Whether or not this is a day on which a month or period ends. (1/0).
day_name	The day as a native date type.
day_name_char	Date (as a string). For example, Apr 02 1995.
day_name_char_weekday	Date (as a string) with weekday prefix. For example, Sun Apr 02 1995.
day_number_in_cq	The number of this day in the calendar quarter, starting at 1.
day_number_in_cy	The number of this day in the calendar year, starting at 1.
day_number_in_fq	The number of this day in the fiscal quarter, starting at 1.
day_number_in_fy	The number of this day in the fiscal year, starting at 1.
day_number_in_month	The number of this day in the month or period, starting at 1.
day_number_in_week	The number of this day in the week, starting at 1.
day_number_til_end_cq	The number of days until the end of the calendar quarter, ending with 1.
day_number_til_end_fq	The number of days until the end of the fiscal quarter, ending with 1.
fq_and_fy_name	Fiscal quarter and year name. For example, Q1 1999.

Table 10: Date Dimension Fields

fq_name	Fiscal quarter name. For example, Q1.
fy_name	Fiscal year name. For example, 1999.
month_and_cy_name	The name of the month abbreviated and the calendar year. For example, Apr 1999.
month_and_fy_name	Month (abbreviated) or period name and fiscal year. For example, Apr 1999.
month_name	The three-letter abbreviations of the month (without a year or any other value). For example, Apr.
month_number	Month or period number since the first date in the system, starting at 1.
month_number_in_cq	Month number since the start of the calendar quarter, starting at 1.
month_number_in_cy	Month number since the start of the calendar year, starting at 1.
month_number_in_fq	Month or period number since the start of the fiscal quarter, starting at 1.
month_number_in_fy	Month or period number since the start of the fiscal year, starting at 1.
month_number_til_end_cy	Number of months or periods until the end of the calendar year, ending with 1.
month_number_til_end_fy	Number of months or periods until the end of the fiscal year, ending with 1.
vacation_day	Whether or not this day is a vacation. (1/0).
week_friday	Date (as a string) for the Friday of the current week. For example, Apr 01 1994.
week_monday	Date (as a string) for the Monday of the current week. For example, Apr 01 1996.
week_number	Week number since the first date in the system, starting at 1.

Date Dimension Fields

Table 10: Date Dimension Fields

week_number_cq	Week number since the start of the calendar quarter, starting at 1.
week_number_cy	Week number since the start of the calendar year, starting at 1.
week_number_fq	Week number since the start of the fiscal quarter, starting at 1.
week_number_fy	Week number since the start of the fiscal year, starting at 1.
week_number_til_end_cq	Week number until the end of the calendar quarter, ending with 1.
week_number_til_end_cy	Number of weeks until the end of the calendar year, ending with 1.
week_number_til_end_fq	Week number until the end of the fiscal quarter, ending with 1.
week_number_til_end_fy	Number of weeks until the end of the fiscal year, ending with 1.
week_saturday	Date (as a string) for the Saturday of the current week. For example, Apr 01 1995.
week_sunday	Date (as a string) for the Sunday of the current week. For example, Apr 02 1995.
weekday	Weekday prefix: for example, Sun.
month_name	The three-letter abbreviations of the month (without any year or any other value).

APPENDIX D

PHYSICAL TYPE VALUES

The tables in this appendix define the database type translations for the physical types you select in EpiCenter Manager's Base Dimension and External Tables dialog boxes. The physical type you select is replaced by its associated database type. The translation of physical type to database type depends on your RDBMS platform.

Notes to sites that use their own database types and SQL macros:

The various fact semantic types (such as Transactional/Statelike) use SUM() operators over the fact columns. New tables are created via SELECT INTO from the result.

With decimal data types such as NUMBER(16,4) that logically map to a NUMERIC(16,4) SQL Server datatype, repeated SUM operations within SQL Server may cause an automatic increase in the column precision, ultimately resulting in a NUMERIC(38) field, which occupies 17 bytes per record. For this reason, both FACTMONEY and FACTQTY map to the MONEY database type in SQL Server. This database type occupies only 8 bytes per record.

Physical Type Values

Table 11: SQL Server Physical Type Values

Physical Type	Database Type
CHAR_1	CHAR(1)
EPIINT	INT
EPIKEY	INT
FACTMONEY	MONEY
FACTQTY	MONEY
IDENTITY	INT IDENTITY
SMALLDATE	SMALLDATETIME
SMALLINT	SMALLINT
SSKEY	VARCHAR(50)
TINYINT	TINYINT
VARCHAR_5	VARCHAR(5)
VARCHAR_15	VARCHAR(15)
VARCHAR_25	VARCHAR(25)
VARCHAR_50	VARCHAR(50)
VARCHAR_100	VARCHAR(100)
VARCHAR_255	VARCHAR(255)

Table 12: Oracle Physical Type Values

Physical Type	Database Type
CHAR_1	CHAR(1)
EPIINT	NUMBER(10)
EPIKEY	NUMBER(9)
FACTMONEY	NUMBER(19,4)
FACTQTY	NUMBER(19,4)
IDENTITY	NUMBER(10)
SMALLDATE	DATE
SMALLINT	SMALLINT
SSKEY	VARCHAR2(50)
TINYINT	SMALLINT
VARCHAR_5	VARCHAR2(5)
VARCHAR_15	VARCHAR2(15)
VARCHAR_25	VARCHAR2(25)
VARCHAR_50	VARCHAR2(50)
VARCHAR_100	VARCHAR2(100)
VARCHAR_255	VARCHAR2(255)

Physical Type Values

09625918 072500

WRITING STAGING SQL STATEMENTS

The Epiphany extraction process (excluding aggregation) consists of two phases: 1) loading data from source systems into the Epiphany staging tables, and 2) running semantic instances against these staging tables. This appendix describes the recommended practices for writing SQL statements that populate the dimension and fact staging tables.

BASE DIMENSION STAGING SQL STATEMENTS

Base dimensions describe the physical dimension tables in EpiMart. You can use EpiCenter Manager to configure the dimension columns for each base dimension table. During an extraction job, one or more SQL statements can be executed against the base dimension staging table to populate these columns.

The purpose of the base dimension staging query is to extract the latest values from the source system. You need not be concerned with determining when a value has changed because Epiphany's semantic templates perform this task.

Base Dimension Staging SQL Statements

You can use the Template feature in the SQL Statement dialog box (see Figure 52. on page 185) to provide an SQL template for a given dimension table. For example, assume you define a base dimension called Customer with the following dimension columns:

- FullName
- Age
- City
- State
- ZipCode

In the Tables References panel of the SQL Statement dialog box, select the option *Populates a dimension*. Choose the dimension table (Customer) from the drop-down list. Clicking the Template button will display the following SQL in the dialog box:

```
SELECT
    <YOUR EXPRESSION> customer_skey,
    <YOUR EXPRESSION> data_modified,
    <YOUR EXPRESSION> FullName,
    <YOUR EXPRESSION> Age,
    <YOUR EXPRESSION> City,
    <YOUR EXPRESSION> State,
    <YOUR EXPRESSION> ZipCode
FROM
    <YOUR TABLE>
```

This is a regular SQL statement for which you must provide the values **<YOUR EXPRESSION>** and **<YOUR TABLE>**. You must assign each column in the SELECT list a valid SQL expression for the value of its destination column. A FROM clause is required to make this a valid statement; a WHERE clause is optional.

The first two columns, *customer_sskey* and *date_modified*, were not specified in the definition of the Customer base dimension. These columns are implicitly added to the base dimension table by Epiphany's Adaptive Schema Generator and must be populated by the staging SQL. The first column is a source system key (*sskey*) and should be unique for every row in the staging table. The concept of *sskey* is important in EpiMart because it tells the semantic templates whenever a row in the staging table represents the same source system entity as a row that already exists in the dimension table. The *sskey* is a variable length string, normally of maximum length 50. It corresponds to the primary key of the source system table or the tables that make up this query.

Note: If the *sskey* column is of interest to end users for querying, then a dimension column populated with the same values will need to be created because the *sskey* is not available for ticket configuration.

The other implicit column, *date_modified*, has the same name in all base dimension staging tables and is used to identify when a base dimension row is inserted into the EpiMart. If the source system contains a *creation date* field, then this field should be used. Otherwise, you can use the source system's expression for "right now," which causes newly extracted rows to assume the date when they were extracted into EpiMart; for Microsoft SQL Server this expression is `GetDate()`. For best results, dates should be returned as strings, for example, 5/26/1999.

The remainder of the columns in the SELECT list must be populated with an appropriate expression for the meaning of that column. Any SQL expression that can be executed against the source RDBMS is valid. Also, Epiphany provides a set of SQL macros that will be automatically expanded to the correct syntax for your source system. Use of macros facilitates the cross-platform usage of your SQL statements. See Appendix A, "Epiphany Macros" for more information.

Duplicate *sskey*'s

An important point about these expressions is that null values are *not* allowed in any field of the staging table. The reason for this is simple: the Epiphany system uses GROUP BY statements at end-user query time to form the tables and charts of front-end applications such as Clarity. However, fact rows that aggregate on columns with null values are left out of the resulting reports because nulls are removed from GROUP BY's. Rather than incurring the query-time penalty for this check, EpiMart insists on non-null dimension column values.

To circumvent this problem, substitute the string UNKNOWN for any null values using the NVL macro. The Epiphany system will automatically generate an UNKNOWN row in your dimension table. The UNKNOWN value is configurable: if UNKNOWN is a valid value in your dimension table, use another value.

Duplicate *sskey*'s

If during a single extraction, a staging table is loaded with two or more rows with the same *sskey*, then the last row entered is used. See Appendix F, "Semantic Types" for a description of the dimension semantic types.

Dimension Staging Queries with Joins

The Epiphany system allows the use of joins in base dimension staging queries. Star schemas typically de-normalize data structures in transactional systems into flat hierarchies, and you must be aware of what the granularity of a base dimension represents in this circumstance.

For example, you will rarely want to use a Cartesian product of two tables in a base dimension staging query, unless the *sskey* of the result set will combine the primary keys of the two tables that are being crossed. It is more common for a single table to "drive" the result set, with other tables joined through unique key lookups to provide additional textual values. For instance, a Product Master table in the source system might represent the driving table of a Product base dimension (with the *sskey* taken from the primary key of the Product Master table), but other tables with textual values for Product Line or Platform may be joined with this master table. In this case, you should ensure that the joined columns of the lookup tables are properly indexed (usually with UNIQUE indexes).

Constructing Base Dimension Queries with DISTINCT Fact Values

Sometimes dimensions are created for which no corresponding master table exists in the source system. For instance, an Order fact may have an Order type associated with it (with several possible choices). These values will be embedded directly in the fact rows on the source, but no lookup table exists with all the choices. In this case, a SELECT DISTINCT query against the source system's fact table might be appropriate for populating base dimension staging tables in EpiMart. The alternative to this method is the use of degenerate dimensions in the fact table, although degenerate dimensions cannot be aggregated.

FACT STAGING SQL STATEMENTS

SQL statements that populate fact staging tables are generally more complex than the ones used to load dimension staging tables. As with base dimension tables, the columns of the SELECT statements are determined by the metadata definition of the fact table (and its constellation) along with certain implicit rules.

To illustrate this point, assume that you define an Order fact in a Sales constellation with the following dimension roles:

- CustomerBillTo
- Product
- CustomerShipTo
- SalesPerson

The constellation contains a single degenerate dimension called OrderNumber, and the Order table has two fact columns: *net_price* and *number_units* that represent the extended amount for an order line item, along with the quantity.

Clicking the Template button on the SQL Statement dialog box (with the *Populates fact table* option selected and the Order table selected in the drop-down list) displays the following SQL in the dialog box:

Fact Staging SQL Statements

```

SELECT
    <YOUR EXPRESSION> ss_key,
    <YOUR EXPRESSION> date_key,
    <YOUR EXPRESSION> transtype_key,
    <YOUR EXPRESSION> process_key,
    <YOUR EXPRESSION> customerbillto_sskey,
    <YOUR EXPRESSION> product_sskey,
    <YOUR EXPRESSION> customershipto_sskey,
    <YOUR EXPRESSION> salesperson_sskey,
    <YOUR EXPRESSION> ordernumber_key,
    <YOUR EXPRESSION> net_price,
    <YOUR EXPRESSION> number_units
FROM
    <YOUR TABLE>

```

As with base dimension staging queries, you must identify what a row in this fact table represents. Based on the columns in this example, a row seems to indicate a line item of a sales order. (In this case, the assumption is that the salesperson gets full credit for a line item; another interpretation of this fact row might be a particular amount of credit that a salesperson received for an order line item.) Typically, the FROM clause of this query would join the Order Line Item table to the Order Header table in the source system.

The columns in the SELECT list can now be divided into these categories:

- Implicit columns that were added automatically
- Dimension role foreign keys
- Degenerate dimension keys
- Fact numeric columns

First, consider implicit columns. As with base dimensions, each fact staging row contains an *ss_key* (notice the difference in spelling) that uniquely identifies this row in the source system. In this example, the *ss_key* might be a concatenation of the Order Number with the Order Line Number (since this combination is presumably unique). *ss_key*'s will be used on subsequent extractions to prevent duplicate copies of the fact row from being created in EpiMart.

The *date_key* indicates when the fact occurred. Since time is a central component of EpiMart, each fact table must contain this column. Many facts are time based; in this example, *date_key* represents the time when the order was placed. However, if time is not important for this fact, then the current system time can be used as a placeholder. Note that *date_key* is granular only to that single *day* when the fact occurred. For best results, the fact SQL Statement should return the day as a string, for instance, 5/1/1999.

Transaction type is another central concept of fact table processing in EpiMart. The SQL statement should return a numeric key that matches with one of the transaction types defined on the Configuration dialog box in EpiCenter Manager. (See Appendix F, "Semantic Types" for more information about *transtype_key*.)

The *process_key* identifies rows from the fact table to be processed by a specific semantic type. (A fact table can contain different types of rows, requiring different semantic types.) For example, the Order fact might hold both Bookings and Shippings, and *process_key* would identify which fact staging rows were which. (See Appendix E, "Writing Staging SQL Statements" for more information about *process_key*.)

Next, you must enter values for each of the dimension role foreign keys. Notice that the names of the columns in the SQL template are *DimRoleName_sskey*. These fields refer back to *sskey*'s of the base dimension tables for this fact. You need to understand the meaning of each base dimension table to ensure that the keys resolve properly. If the *sskey* of the Product base dimension is taken from a Product Master list in the source system, then *product_sskey* in the Order table must also refer to an entry in the Product Master. If a base dimension is the cross-product of two source system tables, the fact staging keys for that dimension must also represent a unique cross-product entry.

Using External Tables as Inputs to Staging Queries

Degenerate keys in the fact staging query should be populated with string values. In the example above, the *ordernumber_key* field would probably be populated with the actual primary key of the Order Header table, such as Order Number 253AD56.

Finally, the numeric columns represent the actual quantities and raw amounts that are associated with each fact entry. Each column should be an additive amount for correct front-end query results. For instance, total dollar amounts for a line item should be populated instead of unit prices because unit prices cannot be added across fact rows.

USING EXTERNAL TABLES AS INPUTS TO STAGING QUERIES

Sometimes it may be necessary to bring data into EpiMart external (temporary) tables before performing any joins; for example, if the source system's SQL limits your ability to manipulate the data. The full power of EpiMart's RDBMS engine can then be used to load the staging tables. In this case, the following sequence of actions is usually employed:

- Step 1:** Drop any indexes on the external tables for fast loading.
- Step 2:** Load the external tables from the source system.
- Step 3:** Create any indexes on external tables needed for fast joins.
- Step 4:** Load the staging tables using queries against the EpiMart External tables. All query plans should use the indexes built in Step 3.

APPENDIX F

SEMANTIC TYPES

This appendix describes the dimension and fact semantic types.

Note: Fact rows with all zero facts are discarded by all semantics.

DIMENSION SEMANTIC TYPES

The dimension semantic types are Slowly Changing Dimensions, Latest Dimension Value, First Dimension Value, and Initial Dimension Load.

Slowly Changing Dimensions

A Slowly Changing Dimension is a dimension in which the attributes or hierarchy of the dimension can change over time, but historical data is not restated. Two examples follow:

- A sporting goods chain decides to rename a store. By using the Slowly Changing Dimensions semantic type, the old name is retained for all of the historical data. One can still identify when the store changed names and compare sales before and after the name change.

Slowly Changing Dimensions

- This same chain is national and has stores divided into three regions. On January 1, 1998, the chain reorganizes its regions, moving Denver from the Central to the Western region. Their 1998 sales forecasts take into account that the Denver store is in the Western region, but they do not want to recalculate forecasts and actuals from previous years. Using the Slowly Changing Dimensions semantic type, sales for Denver can be aggregated up to the Central region through 1997. Beginning January 1, 1998, Denver sales are applied to the Western Region.

The Slowly Changing Dimensions semantic type accomplishes the following logic:

- Rows with the same *sskey* in the staging table will be eliminated following in a “last in wins” rule. This is determined by the special column called *key*, which is created by EpiChannel and is automatically incremented during normal extractions. The highest *key* row for a given *sskey* will be accepted.
- New rows are created by searching through the dimension staging table for new *sskey* values, or *sskey* values that have one or more dimension column changes from the last known values. Each of these cases creates a new row in the dimension table. The mapping row for that *sskey* points to the latest dimension row with that *sskey* value.
- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of changed rows is reported in the *Modified* column. The number of rows in the staging table is reported in the *Processed* column.
- The dimension table has its key column made into a Primary Key index. Each dimension column is also Indexed (non-uniquely). The Mapping table is indexed (primary key) on *iss*, *sskey*. (You can use the Dimension Column dialog box in EpiCenter Manager to disable the indexing of individual columns.)

iss is used when two source systems have the same *sskey* values (such as when the SAP and Vantive systems both have a Customer #2 record). *iss* is determined by the source system identifier selected using the General tab of the Data Store dialog box (Figure 49, on page 178).

Latest Dimension Value

- The column *dimension_key_REAL* (where *dimension* is the dimension column name) is set to the first key value for each *sskey*. In other words, when a new *sskey* is discovered, then the *REAL* key is set to the new dimension key value. Subsequent dimension rows for this *sskey* will retain the original *REAL* key value.
- The UNKNOWN *sskey* always maps to dimension key value 1 in the Mapping table.

Note the following:

- Do not allow dimension column values to *oscillate* unpredictably. (See “First Dimension Value” on page 314 for more information.) In particular, do not rely on *key* filtering of duplicate *sskey* values if two or more rows during a single extraction might have different values for one or more dimension columns. The reason is that a new row will be created in the dimension table for every extraction for which a change is recorded. This can cause two values to “compete” with each other, forcing an unending sequence of row creation in the dimension table.
- Rows can be removed from dimension tables after they have been extracted with an explicit delete or truncation, or through use of the Initial Load Dimension.

Latest Dimension Value

The Latest Dimension Value updates rows instead of performing Slowly Changing Dimensions. It applies changes retroactively to a dimension. Thus the changes take effect for all historical data, as well as for current and future loads.

For example, assume that a sporting goods store has a category historically called *Rollerblades*. Now that they are selling other brands, the store wants to change the category to *In-line Skates*. By using the Latest Dimension Value semantic type, this change can affect all of the historical data because all previous sales of *Rollerblades* are now labeled *In-line Skates*. As a result, the store can compare year-to-year sales of all in-line skates.

First Dimension Value

This semantic type has the same duplicate *sskey* filtering as Slowly Changing Dimensions. Use this semantic type for an implementation that “restates history” when a source dimension table changes.

Note the following:

- New *sskeys* are inserted in both the dimension table and mapping table. Existing *sskeys* with one or more changed dimension columns are updated in place in the dimension table (that is, the same dimension row is used) with the latest values.
- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of changed rows is reported in the *Modified* column. The number of rows in the staging table is reported in the *Processed* column.
- Latest Dimension Value has the same indexing as Slowly Changing Dimensions.
- The REAL column is always equal to the dimension key.
- Latest Dimension Value has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.

First Dimension Value

The First Dimension Value semantic type ignores any changes to a dimension. The values that were present when the row was first inserted are preserved forever, regardless of any future changes. This semantic never modifies a row after it has been seen.

You might want to use this type if your source data comes from two systems that are not in complete agreement with each other. For instance, if one system has Customer #12345 as *David Anderson*, and the other has the same customer as *David Andersen*. Ideally, you would determine which one was in error and correct it.

In the meantime, you could choose to apply the first value read and to ignore the other. (This is a good method for avoiding the *oscillation* problem mentioned on page 313.) If you were to use the Slowly Changing Dimensions semantic type in this case, there would be a race between the two source systems for each extraction, and (in the worst case), your dimension values could alternate between the two values with every extraction.

Note the following:

- First Dimension Value has the same duplicate *sskey* filtering as Slowly Changing Dimensions.
- New *sskeys* are inserted in both the dimension table and the mapping table. Existing *sskeys* are ignored.
- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of rows in the staging table is reported in the *Processed* column.
- First Dimension Value has the same indexing as Slowly Changing Dimensions.
- The REAL column is always equal to the dimension key.
- First Dimension Value has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.

Initial Dimension Load

The Initial Load Dimension semantic type ignores the current Datamart entries. (It is also the fastest semantic type).

Initial Dimension Load loads dimension without regard to any previously existing rows. This can be used for the initial load of the empty EpiMart, and also to completely reload a dimension, ignoring existing values. Using any other semantic type would require emptying the existing dimension table before beginning the extraction.

Note the following:

- Initial Dimension Load has the same duplicate *sskey* filtering as Slowly Changing Dimensions.
- The existing dimension and mapping tables are ignored; all *sskeys* are imported directly into the dimension and the mapping tables.

Fact Semantic Types

- In the EpiChannel log file, new *sskey* rows are reported in the *Inserted* column. The number of rows in the staging table is reported in the *Processed* column.
- Initial Dimension Load has the same indexing as Slowly Changing Dimensions.
- The *REAL* column is always equal to the dimension key.
- Initial Dimension Load has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.

FACT SEMANTIC TYPES

The fact semantic types are Count Unjoined (which replaced the fact semantic type Update Unjoined in Release 3.2), Custom Fact Index, Transactional, Transactional/State-like, Transactional/State-like/Force Close, Pipelined, Initial Load Fact, and Reload Max Date.

Count Unjoined (Optional)

Count Unjoined informs you of the number of rows that have been transformed by an outer join. Outer joins map fact staging table to dimension tables. This semantic type's usage of outer joins always prevents the loss of rows.

As of Release 3.3, Update Unjoined was renamed Count Unjoined because it no longer performs any updating. UpdateUnjoined searched for fact staging entries (in the dimension *sskey* columns) that did not map to any values in the corresponding dimension mapping table. It then set all unmapped values to the special token UNKNOWN. This was necessary because all of the other semantic types, such as Transactional and Pipelined, performed inner joins between the fact and dimension staging tables, which would cause a loss of data if this updating had not occurred earlier.

Releases 3.3 and later employ outer joins when mapping the fact staging table to the dimension tables. This usage of outer joins always prevents the loss of rows, whether or not Count Unjoined has been run previously.

Custom Fact Index

In EpiCenter Manager, you use the Custom Index tab of the Fact Table dialog box to define custom indexes for a fact table. The semantic type Custom Fact Index, however, must be executed for the fact table as part of the scheduled extraction process in order for these indexes to be built.

Important: Schedule the Custom Fact Index semantic instance *after* all other semantic instances for a fact table.

For a discussion of custom indexing, see “Custom Fact Indexing” on page 62.

Transactional

The Transactional semantic type is the simplest means of moving fact staging data into fact base tables. This semantic type uses the following logic:

- Only rows with *process_key* set to 1 (Transactional) are used; others are discarded.
- For *sskeys* that are already entered in the current fact base table, all rows in the staging table with that *sskey* and with the same or earlier *date_key* are discarded. If the *sskey* already exists in the fact table, only later dates can be added to the fact table.

Two or more rows with the same *sskey* can be imported into the fact base table if they arrive in the same extraction and the *sskey* either does not already exist, or exists but is dated earlier. (This property is used by the *pseudo-order* approach to the Booking/Shipping problem; see “Transactional/State-like/Force Close” on page 319.)

- In the EpiChannel log file, all rows added to the fact base table are reported in the *Inserted* column, and the total number of rows in the staging table is reported in the *Processed* column.

Note the following:

- Transactional semantics should be used for event facts; once the fact event has occurred, it can never be modified.

Transactional/State-like

- To reload transactions after they have been loaded into the fact base table, the rows must be deleted from the fact base table, or the fact base table must be truncated. See “Reload Max Date” and “Initial Load Fact” for more information.

Transactional/State-like

The Transactional/State-like semantic type allows changes to already existing rows in EpiMart. It uses the same logic as the Transactional semantic type, with the addition of the logic described below.

First these three steps occur:

- Step 1:** Records in the staging table with *process_key* of 2 (state-like) are treated as Orders that can be *differenced* between extractions (a discussion of this follows).
- Step 2:** For records in which *process_key* = 2, duplicate *sskeys* with the same *date_key* in the same extraction cause only the highest *key* value to be used. Other rows are discarded. This is the same filtering that happens in dimension semantics such as Slowly Changing Dimensions.
- Step 3:** For an *sskey* with the highest *key* (which means that for every set of rows with the same *sskey*, take the row with the highest *key*), if the *date_key* for that staging row is less than the last *date_key* for that *sskey* in the fact base table, the staging row is discarded. In other words, an Order can only be modified on its last reported date, or some time further into the future.

After Steps 1-3, the staging fact columns and dimension values are compared with the current values in the fact table (if any). Adjusting records are created in the fact table so that the fact table now reflects the reported *state* from the staging table. (This is why the term *state-like* is used.) *Differenced* transactions are invented if the numeric fact columns have changed.

If the dimensionality has changed, then the Order will be “de-booked” and “re-booked” with the correct dimensionality.

If the same *sskey* appears in the staging table with more than one *date_key*, then further adjusting transactions are made in the fact base table as appropriate to bring the fact base table in line with the reported staging rows.

Note: By convention, Bookings are entered with positive facts (negative for Returned Orders), whereas Shipments are entered with negative facts (positive for Returned shipments).

When using this semantic type it is difficult to ensure that Backlog calculations will remain consistent when Orders are not completely closed in the source system. Use the Transactional/State-like/Force Close instead.

Transactional/State-like/Force Close

This semantic type is equivalent to Transactional/State-like with the following additional logic.

Once a Booked Order is entered into EpiMart, it remains in that state (with an Open Backlog) forever. In normal scenarios, an invoice eventually arrives, which will close the Backlog. However, in some source systems, Booked Orders can be removed from the system completely. If such an Order had been entered previously into EpiMart, then it will remain in an Open Backlog condition forever.

The solution to this problem is to use Transactional/State-like/Force Close, which establishes a “Challenge Protocol” for Open Orders. In this scenario, all Open Bookings must be extracted into the staging table during every extraction. The reason is that the Force Close logic will close out all Open Orders (*sskeys* with non-zero facts in the system) that do not appear in the fact staging table. Only Booking Transaction types (those with *transtype_key* values between 1 and 99) will be affected in this manner.

When using this semantic type, the methodology that has been found to work in practice involves the use of pseudo-orders as follows:

- One extraction SQL statement extracts all Open Orders. Fact amounts are the Open amounts (what has not been shipped yet), not the ordered amounts. Transaction types for this statement should be in the Booking range (1...99), and *process_key* should be 2 (state-like).

Pipelined

- The second extraction statement uses *process_key* of 1 (Transactional) and represents all shipments in the system. These records normally go into the system with negative facts for positive shipments (by convention). These transaction types should be in the shipment range (101 or greater).
- The third statement is a restatement of the shipment, but as a Booking (*transtype_key* between 1...99). The *process_key* is still 1 (allowing the Transactional Semantics to import it), but the transaction type is a Booking. The same *sskey* and dimensionality as the second statement are used. These are the pseudo-orders since they are actual shipments that are entered as Orders.

The net effect of this methodology is that as a shipment is reported against Bookings, the Open Booking quantity in statement 1 is decremented, while the actual shipment is restated as a Booking transaction. Eventually, when the Order is removed from the system, the Force Close logic will close out any remaining Open fact quantities from statement 1 above. What remains in the system will be Shipments and their corresponding pseudo-orders. By construction, the Backlog will be zero.

Pipelined

Use the Pipelined semantic type for facts that can exist in several different life-cycle phases, called pipeline states; for example, sales opportunities or support calls facts.

The *transtype_key* field represents the pipeline. Before using this semantic type, you must define the pipeline stages and numbers.

Pipelined generates these transactions:

- Step 1:** When an *sskey* first enters a particular pipeline stage (*transtype_key*), then a positive fact row is created with that *transtype_key*.
- Step 2:** If that *sskey* subsequently moves backwards in the pipeline, then in addition to the new row created by the previous step for the new pipeline stage, a negative “de-booking” transaction is created with the old *transtype_key* minus 1. (This is a loss transaction.)

09625518.072500

Initial Load Fact

Step 3: If that *sskey* subsequently moves forwards in the pipeline, then in addition to the new row created in Step 1 for the new pipeline stage, a negative “de-booking” transaction is created with the old *transtype_key* plus 1. (This is a shipment transaction.)

Note: As a consequence of the +1 and -1 rows, the pipeline should be designed so that the *transtype_keys* that represents bookings in various stages are at separated by at least 3 integers.

For example, assume that the pipeline consists of the following steps: Prospect, Lead, Quote, and Order. You could set up *transtype_key* fields for this pipeline as follows: 302 = Prospect, 305 = Lead, 308 = Quote, and 311 = Order. The semantic type creates its own transactions for 301, 303, 304, 306, 307, 309, 310, and 311, which correspond to the forward and backward movements from the various pipeline stages.

To define measures that extract information such as Number of New Leads, you must define transaction types with these new names and keys (using EpiCenter Manager’s Configuration dialog box; see “Transaction Types” on page 288.)

Note: The pipelined semantic types ignores the *process_key* field because it is no longer relevant to it.

Initial Load Fact

Similar to the Initial Dimension Load, the Initial Load Fact semantic type is the fastest way to load a fact table. It also has the special property that the current table (whether A or B) is ignored so that this semantic type can reload an already-populated fact table. All existing rows, however, are lost. For this reason, Initial Load Fact is usually used during development when an installation is verifying whether a first extraction yields correct data.

006255318.072500

Reload Max Date

Important: For first time extractions, you can use this semantic template in place of all other fact semantic types that load data when each *sskey* is being loaded into the staging table *only once*. This is because upon first extract, when an *sskey* is loaded only once, the special logic of the other semantic types (such as delta inference during Transactional/State-like) does not apply. If the first extraction does require multiple records per *sskey*, such as loading inventory values using Transactional/Statelike, then Initial Fact Load cannot be used.

Reload Max Date

The Reload Max Date fact semantic type is a hybrid of the Transactional and Initial Fact Load semantic types. Use it to reload a subset of an existing table in which the reload point is separable by date only.

Reload Max Date first determines the minimum date that occurs in the fact staging table. It then copies from the current fact table to the new fact table all existing EpiMart data that occurs with a date key prior to that minimum date.

The fact staging table is then used to hold all subsequent dates. Thus a complete load of data must be placed in the fact staging data from the minimum date forward. When loading the fact staging table (via SQL statements), you should use a WHERE clause based on a particular date.

EXPORT/IMPORT OF METADATA

All of the control information for an EpiCenter is stored in a single metadata repository called EpiMeta. EpiMeta represents a transactional, fully relational model of over one hundred and fifty tables, with complicated declarative referential integrity constraints. Epiphany provides EpiCenter Manager for configuring this metadata without the need to write to, or even know about, the underlying data structures.

Epiphany also provides a metadata Export/Import utility for moving metadata between EpiCenters and for backing up the definition of an EpiCenter. To use this tool properly, you should understand the basic metadata concepts discussed in this appendix.

METADATA OVERVIEW

All of the user-configurable metadata tables have integer primary keys. These non-natural keys are provided by the database engine when a metadata row is inserted; the value of the primary key itself has no intrinsic meaning. All relationships to this inserted row are made via this integer. For instance, the relationship between a filter group and its filter block is represented in EpiMeta via an integer column called *filter_block_key*.

Metadata Overview

The Access Export database does not simply contain a copy of the metadata tables being exported for this reason: If data were copied to the Export file, then when this same information is imported into a new EpiMeta, the integer primary key values in the Export file might clash with already existing primary keys in the target EpiMeta. Therefore, the Access database uses an EpiMeta-independent representation of metadata. See “Export File Format” on page 326 for a description of these Access database tables.

EpiMeta contains many tables, all of which are inter-related. Ticksheet metadata refers to constellation metadata (for instance, the attributes refer to dimension columns), while security metadata refers to ticksheets. In order to export only a portion of the metadata at a time, the Export machinery must decide where to stop exporting—otherwise, the entire metadata must be exported with each operation.

When using the Export metadata command of EpiCenter Manager, you must select which part of the metadata to export. Figure 83 shows an overview of the various domains of metadata within EpiMeta.

Each rectangle in the figure represents one of the options available for export. Everything within the rectangle is actually contained as metadata in the Export file when that option is selected. The arrows in the diagram represent references to other metadata. These references are contained in the Export file as well, but the references are made to other objects by name only. In other words, when exporting ticksheets, the measure mapping metadata is exported in the Export file, but the measures themselves are not exported (unless you also select the Measure option). Only a reference to the appropriate measures is exported.

Upon import, these references are used as follows. When ticksheet metadata is imported into a different EpiMeta, the import machinery searches for measures with the referenced names. If it finds these, then the same relationships are established in the new EpiMeta as the ones that were exported. However, if the Import machinery does not find these measures by name, the measure mapping information will be lost upon import.

Replacing Existing Metadata on Import

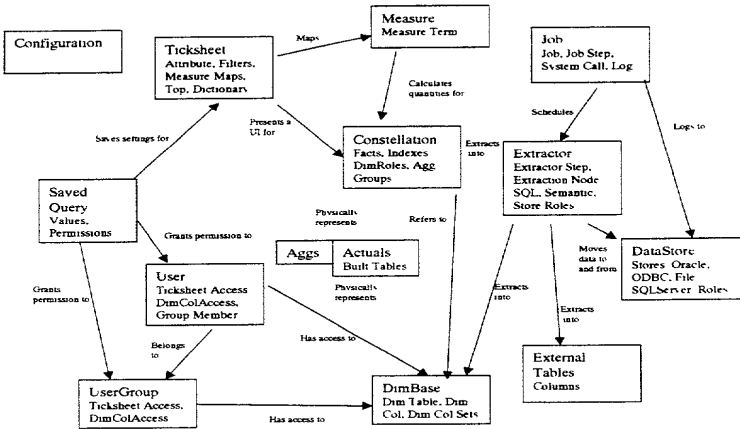


Figure 83: EpiCenter Data Model

Replacing Existing Metadata on Import

When importing metadata into an existing EpiMeta, the Import machinery will detect an attempt to overwrite existing metadata. Existing top-level objects are overwritten in place. Their children are deleted, however, before being recreated. The definition of “existing” is usually based on a unique name column for one of the metadata tables. For instance, ticksheets must have unique names, so an attempt to import a ticksheet with the same name as an existing one results in a warning message (unless the Always Replace option has been selected previously).

Actuals and Agg Metadata

The Export All command in EpiCenter Manager does not export the entire contents of EpiMeta. The options for Actuals and Aggs are omitted by default from this export. This is because these sections of metadata are “derived” metadata: the Adaptive Schema Generator produces Actuals metadata, and Aggbuilder produces Agg metadata. As a result, a new EpiCenter can be constructed using the Export All metadata by rebuilding the schema, re-extracting, and re-running Aggbuilder.

To “clone” an EpiMeta in such a way that EpiMart itself will not need any modification when it is used with this new EpiMeta, you must select the options for Actuals and Aggs.

Note: An Export All operation on a running system, followed by a re-import of that metadata causes all Actual and Agg metadata to be lost.

Export File Format

Each Microsoft Access Export database has the same schema. In fact, this schema can be thought of as a meta-schema for representing relational data. For a description of these tables, see Table 13, “Export Tables,” on page 327.

09625518 072500

To modify the row contents contained in an Export file, edit the *Export_col* table, which is simply a collection of name/value pairs for columns.

Table 13: Export Tables

Table Name	Description
Export_tbl	One row per metadata table being exported.
Export_row	One row per metadata row being exported.
Export_col	One row per column per row of metadata being exported. Only non-relationship columns are contained in this table.
Export_rel	One row per relationship between two rows of metadata. Can be a relationship between two rows contained in the Export file, or between one row in the Export file and one reference to a row in a foreign EpiMeta.
Export_status	Header information about the Export file.
Rel_parent	A reference to a metadata row in the foreign EpiMeta.

Export File Format

0905510 072500

TROUBLESHOOTING

This chapter describes the Epiphany error conditions and error messages and suggests action you can take to resolve problems.

If you need additional assistance, please e-mail Epiphany Customer Support: support@epiphany.com.

REGISTRY EDITOR WARNING

Using Registry Editor incorrectly can cause serious problems that may require you to reinstall your operating system. Microsoft cannot guarantee that problems resulting from the incorrect use of Registry Editor can be solved. Use Registry Editor at your own risk.

For information about how to edit the Registry, view the “Changing Keys And Values” Help topic in Registry Editor (**Regedit.exe**), or the “Add and Delete Information in the Registry” and “Edit Registry Data” Help topics in **Regedt32.exe**.

Note that you should back up the Registry before you edit it. If you are running Windows NT, you should also update your Emergency Repair Disk (ERD).

SQL SERVER ERROR MESSAGE

Cannot Connect to the Server

The most likely cause of this problem is that the ODBC driver for SQL Server has not been correctly installed.

You can verify this by opening Start\Settings\Control Panels\ODBC\ and selecting the tab for the ODBC Drivers. Make sure there is an entry for SQL Server. If there is not, you can use the appropriate CD-ROM to reinstall it:

SQL Server 7.0: The ODBC driver is installed with the SQL utilities on the SQL Server CD-ROM.

APPLICATION SERVER ERROR MESSAGES

Note: If the Application Server won't start, you can run the Scrutiny debugging tool in EpiCenter Manager to diagnose the problem. See "Running the Scrutiny Debugging Tool" on page 223.

The following Application Server errors messages are described:

- "Cannot Connect to the Server" on page 330
- "User Cannot Log In" on page 331
- "Service Control Manager Fails to Start Application Service" on page 333
- "Out of Memory" on page 334
- "Invalid Object DATE_O Error" on page 334
- "Not a Valid Application Error" on page 335
- "Internal Windows NT Error" on page 335
- "EpiQuery Engine Database Connection Open Failure Exception" on page 336
- "Charts Do Not Display" on page 337
- "GIF Images Fail to Display on Web Pages" on page 337
- "No Results Available for a Query" on page 338

- “Result Page Error: Extraction Date Unknown” on page 339
- “Web Server Message: Object Not Found” on page 340
- “Browser Crashes When Retrieving Results from Application Server” on page 341
- “Refresh Program Fails” on page 342
- “Application Log Full Error” on page 342
- “Application Server Log Security Problem” on page 343

User Cannot Log In

There are three types of failure associated with an unsuccessful login to the Application Server:

Step 1: The Application Server generates a critical exception indicating an application error, or an error related to the NT security domain).

If this case, an error message appears in the browser with a link to the log that contains the stack trace for the error. Read the error description. Generally, it relates to the way NT domain security is configured or set up at this point. For example, the error might say that the domain controller could not be reached.

Try to fix the problem based on the specifics of the error message. If you need additional assistance, please e-mail Epiphany Customer Support: support@epiphany.com.

Step 2: An invalid login message displays even though there is a valid username and password.

This error may result when the search for the username/password occurs on the wrong machine. For example, suppose a user *foo* existed on both the machine local to the Application Server and the primary domain controller.

The order in which the user *foo* is searched is as follows:

- the local SAM
- the primary domain

User Cannot Log In

- the trusted domains

Thus, user *foo* in the local SAM will be found first, even though the user-name/password combination could have been for the user *foo* that exists in the primary domain or in a trusted domain.

To solve this problem, further specify the username by including the domain name before the username; for example, *EPIPHANY\foo*. In general, specify the full user's name if the user's browser displays an invalid login message.

If the error persists, please e-mail Epiphany Customer Support: support@epiphany.com.

Step 3: Authentication was successful, but the user is not allowed to use the Epiphany system.

For a user to have access to Epiphany System, he or she must be a member of at least one Epiphany group after a sync-up process.

Use EpiCenter Manager to check group memberships after you receive this error message. If the old group memberships were removed, then this error has occurred because in the NT domain, the user is not a member of the NT groups that have the corresponding groups marked Synchronize in EpiCenter Manager.

Also check that the username used to log in matches the username in EpiCenter Manager. If the username in EpiCenter Manager is prefixed with domain name other than the one that the actual NT user is a member of, then the login could fail and return an error message to this effect. Specifying the full username upon log in may fix this problem.

If the error persists, please e-mail Epiphany Customer Support: support@epiphany.com.

Additional Action to Take If User Still Cannot Log In

Note: If you need to have a working system immediately, you may temporarily disable security by hooking up EpiPassThruLogon module. Be sure to read “**Application Server Security**” on page 250 for instructions on configuring authentication modules.

If you cannot fix the problem using the above procedures, e-mail a security log (xxx-xxx-xxx-SECURITY.txt) and Application Server log (xxx-xxx-xxx-SRV) to Epiphany Customer Support.

A description of the three security logs follows:

- APPSERVER.LOG** The only file with a suffix of **.log**, this is the main Application Server activity log, which logs a summary of all queries performed on that server. Includes the user who submitted the query request, the type of ticksheet submitted, the failure or success of the query, and the amount of time for the query.
- SAVE_RESTORE** Logs all save and restore operations during a session.
- QM_randomly_generated_alphabetic_sequence_username** Consists of individual logs, one per user, that show the content of each query or other browser transactions. These logs contain the selected attributes, measurement selections, filters, and so forth submitted with each query. It also contains the SQL sequences, the duration times of the various stages of the query, and the total time for each query.

Service Control Manager Fails to Start Application Service

If after installing an Application Server, the Service Control Manager (Start Menu\Settings\ControlPanel\Services) fails to start the Application Server, then double-click the service in question, and make sure that Allow Service to Interact with Desktop is selected.

Out of Memory

Out of Memory

The Application Server requires sufficient memory (at least 64 megabytes) to function properly. The default **jre** program allows the Application Server to allocate only 16 megabytes of extra memory. Queries that involve Product and Customer (or any large dimension) and have the Rows option set to All can easily exhaust the default 16 megabytes. When this happens, you will receive an Out of Memory exception error: `java.lang.OutOfMemoryError`.

The solution to this problem is to make sure that you have started the Application Server with the `-mx64M` parameter. For example:

```
jre -mx64M -classpath ... com.epiphany.server.Server ...
```

The `-mx64M` parameter allows the Application Server to allocate up to 64 megabytes of memory.

The Refresh program can also exhaust memory if run three times consecutively. This is because the atomic switch from old to new metadata requires a least two copies of the metadata to be alive at the same time. Until all references to the old metadata are released, both objects stay in memory. Eventually, as sessions are cleaned up and as old command threads die, the old object will be released.

VirtualMartDatabase Key Missing Error

If you receive this message from the Application Server, more than likely a valid EpiCenter data store has not been specified.

When you configure an EpiMeta database using EpiCenter Manager, you will use the EpiMart Data Store dialog box in the Extraction\Data Stores folder to specify the target EpiMart. Click the Properties tab and make sure that the data store name is correct.

Invalid Object DATE_O Error

When starting the Application Server, you may encounter the SQL error **Invalid object Date_0**. This error is generated when the Application Server attempts to read the metadata (EpiMeta).

To correct this problem:

- Make sure that the EpiMart data store specified via EpiCenter Manager points to a valid EpiMart database. (Open the Extraction\Data Stores\EpiMart Data Store dialog box, and click the Properties tab.)
- Make sure that you have populated the date dimension in your EpiCenter using EpiCenter Manager. To verify this, go to ISQL and use the **sp_help** command on your EpiMart database. Check for the existence of the *Date_0* table. If it is not there, it has not been populated. See “Getting Started” on page 87 for instructions.

Not a Valid Application Error

This problem can occur for two reasons:

- If a service component required for Windows NT, an application, or a network protocol, is corrupted or missing.
To correct this problem, manually expand the service component file. For example, if the service name in Event ID 7000 is MUP, expand MUP.SY_ from the Window NT CD-ROM to MUP.SYS in the %SystemRoot%\SYSTEM32\DRIVERS folder.
- If the folder location of the executable contains spaces in the directory name (that is, has a long filename). An example would be when the executable is located in the \Program files\service.exe folder.
To correct this problem, modify the Registry key that contains the executable path so that it is enclosed in quotation marks.

Internal Windows NT Error

This error results when the Epiphany Application Server cannot be started as a service. The exact error message that was returned from the Epiphany Application Server is logged in the Windows NT Event Log.

EpiQuery Engine Database Connection Open Failure Exception

To locate this error message in the Event Log:

Step 1: Go to Start menu\Programs\Administrative Tools (Common)\Event Viewer.

Step 2: Click the Log menu and select the Application.

Step 3: Double-click the appropriate event.

All Epiphany Application Server events have the source EpiAppServer.

To solve this problem, first stop any running Application Server services. Then log onto Windows NT as an administrator and re-install the Epiphany software. If this does not solve the problem, start the Application Server from the command line as described in "Running as a Console Application" on page 230. The console output should describe what is wrong.

EpiQuery Engine Database Connection Open Failure Exception

An EpiQueryEngineDBConnOpenFailureException from the EpiQueryEngine means that the Application Server is having difficulty connecting to the database. Take these steps to correct this problem:

- Make sure that the username and password for the EpiMeta database are set correctly in the Windows Registry.
- Make sure that the database name and server are also configured properly (again, in the Registry).
- Make sure that SQL Server TCP/IP Sockets have been enabled. Usually, this option is set when the SQL Server is installed on the machine. To verify that these sockets have been enabled, follow these steps:

Charts Do Not Display

- **SQL Server 7.0:** From the Start menu, open Microsoft SQL Server 7.0 and select Client Network Utility. Select the Network Libraries tab in the SQL Server Client Network Utility dialog box. TCP/IP needs to be configured as client for Epiphany to connect to the SQL Server. To configure TCP/IP:
- Step 1:** Select the General tab in the SQL Server Client Network Utility dialog box.
- Step 2:** Click Add.
- Step 3:** Select TCP/IP in the Add Network Library Configuration dialog box.
- Step 4:** In the Server alias box, enter the alias of the computer that runs SQL Server and listens on the Windows Sockets Net-Library.

You can also specify the server with its IP address instead of its name. See *Microsoft SQL Server 7.0 Books Online* for more information.

Charts Do Not Display

If there is a broken link to an image, or no image appears, or an image with an error message appears instead of the chart, there is a charting problem. Please contact Epiphany Customer Support.

GIF Images Fail to Display on Web Pages

If GIF images do not appear on your Web pages, do the following:

- Step 1:** Make sure that your Web server has an alias for your *instance_name* that points to a valid directory. If you performed a normal installation, then all Web files should be located in the directory:

C:\Program Files\Epiphany\Instance_name\web\WWWROOT

and GIF files should be located in the directory:

C:\Program Files\Epiphany\instance_name\web\WWWROOT\images

(assuming your Epiphany Application Server was installed in C:\Program Files).

No Results Available for a Query

If your Web server is IIS 3.0, you can find the aliases by opening up the IIS Internet Service Manager (normally this is located in your Start\Microsoft Internet Information Server menu) and selecting the Directory tab. Make sure that there is an entry for *instance_name* that has a valid directory.

For the IIS 4.0 Web server, open the Windows NT Option Pack\Microsoft Internet Server\Internet Service Manager. Aliases are listed in the IIS Management Console.

- Step 2:** Make sure the **WWWROOT\images** directory has the GIFs in it. If you are missing a few GIFs, then you will need to reinstall your Application Server or copy the GIFs from a different instance.
- Step 3:** Check the BASE HREF tag that is defined in the source of the page. In your browser, try to view the source for this HTML page. Look for the BASE HREF tag at the top of the page. Note what it is and make sure that it is a valid alias using the procedure above.
- Step 4:** Make sure your Web server is serving pages and that your browser is not displaying cached HTML. Clear the caches (Memory and Disk) on your browser, close your browser, and try to access the URL again. Also, try referencing another URL from your Web server to make sure that it is running.

Technical Note: All of the GIFs on an Application Server-generated page are referenced from the **images** directory, which is relative to the BASE HREF specified at the top of the page in a META tag. The *instance_name* is derived from the URL that you use to access the Application Server. It is used throughout the system to read the correct Registry entries and to generate the correct URLs.

No Results Available for a Query

The most common cause of this problem is that the *current_datamart* value in the EpiMeta database is set to the wrong database.

Result Page Error: Extraction Date Unknown

Use EpiCenter Manager to make sure that the *current_datamart* is set to the correct database. You can also use the ISQL tool to determine which database, A or B, has the most current data.

Result Page Error: Extraction Date Unknown

The *last_extract_date* is a field that is kept in the EpiMeta database. It is used to keep track of the date displayed on the top of all Clarity and Relevance reports as the date of the last extraction. It is normally populated by the extraction SQL entered in the End of Extraction job in the EpiCenter Manager. It can also be populated by EpiCenter Manager via the Configuration dialog box. This field must be entered in one of two very strict formats. The default format is mm/dd/yyyy; for example: 01/14/1998.

The Application Server applies the following logic to parse the date:

Step 1: If the field has more than 10 characters, then parse it using the pattern mm/dd/yyyy hh:mm:ss. Otherwise, use the pattern mm/dd/yyyy.

Step 2: If the parse fails, then use {extraction date unknown}.

In addition, the date that is displayed at the top of the report always has a time zone. The time zone is printed based on the default time zone of the machine on which the Application Server is running. The date that is being displayed is also taken into consideration. For example, the date 12/20/1997 will display as December 20, 1997 PST if the Application Server was running on a machine in California. However, the day 05/12/1998 will display as May 12, 1998 PDT on the same machine since Daylight Savings time took affect in April.

Note: EpiCenter Manager does not allow the user to enter a date in the format mm/dd/yyyy hh:mm:ss. Only the SQL in the extraction job can enter dates of this format, or you can manually arrange this via ISQL.

Web Server Message: Object Not Found

Web Server Message: Object Not Found

If you installed the Epiphany software using the standard Epiphany software installation program, you will access your Web server through this type of URL:
http://machinename/scripts/instance_name/Epiphany.dll.

If you receive an object not found error message, follow these steps:

Step 1: Start and stop the Web server. If this does not solve the problem, go to the next step.

Step 2: Verify the Web server is serving pages.

Note: Make sure that your browser is not just returning cached HTML pages by clearing your memory and disk cache before testing.

Try to access other URLs from the same *machinename*. Try to access other static HTML files that are installed as a part of the Application Server installation, such as **http://machinename/instance_name/clarithelp.html**.

Step 3: If this does not work, try accessing any other file that the Web server should be serving. Consult the Internet Service Manager for the names of other aliases that the Web server should be serving, and then try to access these aliases with your browser.

In most cases, your Web server searches the **C:\Inet-pub\scripts\instance_name** directory to find the **Epiphany.dll** program. Make sure that there is such a directory on your machine, and that the **Epiphany.dll** file is in that directory.

Step 4: Check the file permissions for the **Epiphany.dll** file.

First, make sure that the account IIS uses for anonymous logins has file access permissions for the **Epiphany.dll** file. Go to the IIS 3.0 Internet Service Manager and look at the Anonymous Login account box. In IIS 4.0, right-click the name of the machine and choose Properties. Select the Direc-

Browser Crashes When Retrieving Results from Application Server

tory Security tab. In the Authentication Methods dialog box, select Allow Anonymous Access and click the Edit button to modify the account used for this purpose. Make sure the user and password are correct.

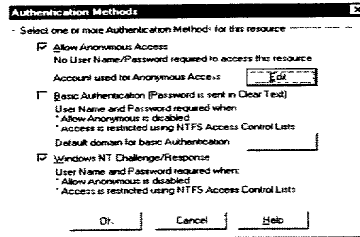


Figure 84: Authentication Methods Dialog Box

To check file permissions, open the Windows NT Explorer window and right-click the **Epiphany.dll** file in the `./scripts/instance_name` directory specified above. Go to Properties and open the Security tab. Click View Permissions and make sure that the account that you used for Anonymous Web Login has access to this file. (If Everyone is selected, then all people, including the Web login account, have access to the file.)

Browser Crashes When Retrieving Results from Application Server

In general, for machines with less than 32 megabytes of RAM, the browser will perform very poorly when parsing HTML files that are larger than 150 kilobytes. Therefore, users who do not have at least 32 megabytes of RAM installed on their machines should refrain from retrieving large queries.

09625518.072500

Refresh Program Fails

An example of a large query follows. Suppose you query Customer by Fiscal Year and apply the filter Business Unit: Learning. This query returns approximately 3800 customers, and the HTML that is generated is 1.8 megabytes. When loaded in Netscape 4.03, it occupies 37 megabytes, takes 3 minutes to parse, and consumes the entire processor. The parsing is the bottleneck in the downloading of the file.

Note: When started, Netscape Navigator 4.03 requires 8 megabytes immediately to load whatever it is loading. Microsoft Internet Explorer (IE) requires 6.7 megabytes, and is substantially faster at parsing the text.

Refresh Program Fails

If the Refresh program fails, the Application Server will continue to use the old metadata information. Users will still be able to log in and view the old ticksheets. This is because the Refresh program reads the new metadata into temporary structures that are atomically exchanged with the old structures only if all of the refresh structures were read correctly.

To solve this problem, rerun the Refresh program.

Application Log Full Error

If you receive this error, you can take the following action to delete all log events.

Note: If you do not want to delete all of the log events, you can also increase the amount of space allocated to the application log. You can also select the Override as Needed option in the Log Properties dialog box.

Step 1: Open Start\Programs\Administrative Tools (Common)\Event Viewer.
Select the Application Log under the Log menu item.

Step 2: Select Clear All Events from the Log menu.

When prompted whether you want to save the log files, and whether you really want to delete all of the log events, respond in the affirmative.

Application Server Log Security Problem

The **APPSERVER** and **SRV** logs contain all of the session IDs that have logged in. Although unlikely, it is possible that someone could alter a session ID (along with a fake IP address) and run queries against the Application Server.

The log file also contains information about who is running a query, and when they are running a query, which may be confidential information. The log files for each query contain all the information necessary to rerun a query. Hence, a malicious party can also determine the type of queries run by a particular user.

It is important to give user access to the log file directory so that when problems occur, the Download query log link at the bottom of each results page will work. Directory browsing for the log file directory, however, should be turned off.

To turn off directory browsing, de-select the Directory Browsing Allowed option in the Directories tab of the WWW Services Properties dialog box in the Internet Service Manager. (Normally, IIS is located in your *Start\Microsoft Internet Information Server* menu.)

09625518.072500

Application Server Log Security Problem

09625518.072500

GLOSSARY

Actual Tables

The Actual tables are metadata created by the Adaptive Schema Generator after it has built or modified tables in EpiMart. The Actual tables are consulted on subsequent schema generations to determine delta operations to perform. Also, other tools examine these tables as a check to ensure that EpiMart's schema matches the current metadata definition.

Adaptive Schema Generator

The Adaptive Schema Generator is a component of EpiCenter Manager that builds the tables in EpiMart using the schema metadata definitions in EpiMeta. When schema metadata changes, this program can perform delta operations to modify the schema without losing data in EpiMart.

Aggregate Builder (Aggbuilder)

Aggbuilder is the executable program (**agg.exe**) that builds aggregate tables in EpiMart. Normally, the execution of this program is scheduled after all data has been extracted from source systems and merged into EpiMart.

Aggregate Group

An aggregate group is a metadata definition of the fact aggregates to be built. It is a series of instructions that tell Aggbuilder which aggregates to build on one or more fact tables. An aggregate group applies to a single constellation. The actual fact aggregates that get built are determined by a combinatorial expansion of the instructions in the aggregate group.

GLOSSARY

Aggregate Navigation

Aggregate navigation is the process by which the Epiphany query machinery determines the optimal aggregate table to use to satisfy an application's request for information. Applications such as Clarity do not need to know about the presence of aggregates because the aggregate navigation process makes the aggregate layer abstract.

Application Server

The Epiphany Application Server is the Windows NT Service that connects with a Web server and serves up Epiphany ticksheets and reports. An Application Server is configured to connect with an EpiCenter.

Attribute

An attribute is a dimension selection on a ticksheet. The row and column lists in Clarity contain attributes. Attributes are related to exactly one dimension column in a table. The only difference between an attribute in a table and one in the Epiphany system is that an Epiphany attribute has an associated display label, such as Fiscal Year, that can be configured via EpiCenter Manager.

In Relevance ticksheets, an end user may select attributes in order to forecast trends, or to analyze highs and lows, best and worst cases, or graphs for various aspects of your business.

The Momentum attributes are derived from the group or individual demographic dimension tables.

Attribute Role

An attribute role is the usage of an attribute on a ticksheet. Attributes are defined only once per ticksheet. For example, if you want a single attribute to appear in both the rows and columns listboxes of Clarity, when configuring the ticksheet, assign that attribute both of these roles: Clarity Row and Clarity Column.

Backlog Type

Backlog type is used on a measure term to specify that the measure term should exhibit accumulation behavior. When time is one of the dimensions of a query, the results for different time periods will exhibit beginning or ending accumulated values instead of the actual transactions that occur in that time period. BEGIN specifies beginning accumulated value, whereas END specifies ending accumulated value.

Base Dimension

A base dimension is a physical dimension table in EpiMart, such as Customer and Product. Base dimension tables contain the actual dimensional values that are available for reporting or filtering. Base dimension tables can be used more than once in a single constellation via dimension roles. Base dimensions are defined for the entire EpiCenter and can be shared by all of the constellations within the EpiCenter.

Base Dimension Aggregate

A base dimension aggregate is a physical table in EpiMart that represents an aggregated view of a base dimension table. A base dimension aggregate results from the removal of one or more columns from the base table, followed by the removal of duplicate rows caused by this deletion.

Base Table

Base tables are the EpiMart tables (both fact and dimension) that store non-aggregated customer data at the level of granularity of extraction. These tables will be used as input to Aggbuilder to build aggregate tables with the same information at different levels of granularity. Base table names end with an underscore zero (_0); for example, Order_0_A or Order_0_B.

GLOSSARY

Constellation

A constellation is a grouping mechanism for like-structured fact tables. A constellation defines the dimensionality of all fact tables in that constellation. Dimension roles and degenerate dimensions are defined for a constellation, not a specific fact table. All fact tables placed in that constellation then inherit that definition. Both ticksheets and measures are defined within the scope of a single constellation, and thus apply only to the dimensionality defined by that constellation.

The Momentum constellation can be associated with multiple Momentum Adjacent Constellations. (See *Momentum Constellation* and *Momentum Adjacent Constellation*).

Custom Index

A custom index is the metadata definition of indexes to build on fact tables in EpiMart. Normally, each fact table is indexed by the leading term *Date*, which works only for queries that are selective by date. You may use EpiCenter Manager to specify additional indexes to build. To actually build these indexes, however, you must apply the semantic template Custom Fact Index.

Data Store

A data store is a logical location of data to be used either as a source or sink within an EpiCenter. Data stores include relational database connections, and files and directories.

Dataset

A dataset is a logical user-interface grouping of ticksheets that display the same view of data. Typically, different ticksheets types (such as Clarity and Relevance Profiling) that show the same data within a constellation are placed within a dataset. The dataset appears to end-users as a named entity that corresponds to a single business process. For instance, a dataset called *Executive Summary* might be defined that shows similar amounts of high-level data, using different Epiphany applications.

Date Dimension

The Date dimension is a special base dimension table supplied by Epiphany, which is used for storing all attributes related to time. All fact tables in an EpiCenter receive the foreign key *date_key* to this table.

Degenerate Dimension

A degenerate dimension is a single column dimension in a fact table that stores textual information in lieu of using a foreign key to a base dimension table. Degenerate dimensions are used when a single field of data fully specifies a dimension of that fact. Examples include Invoice Number and Serial Number.

Because degenerate dimensions appear only in fact base tables, they are never aggregated. (They are, however, always indexed.) Degenerate dimensions are defined on a constellation rather than for a specific fact table. All fact tables within that constellation inherit the degenerate dimension definition.

Dimension Column

A dimension column is a single column or attribute of a base dimension table. It contains the actual customer data that appears on reports or in filters.

Dimension Column Access

Dimension column access is the ability for a user or group to view only a subset of the available data in an EpiCenter. For example, dimension column access settings can be used to limit specific users to only data for the Western Region.

Dimension Column Set

A dimension column set is a named set of dimension columns (all within a single base dimension) that defines which columns will be used in a base dimension aggregate.

GLOSSARY

Dimension Role

Dimension roles allow a single base dimension table to be used in different roles within a constellation; for example, a Territory base dimension table includes Eastern Sales and Western Sales data. A Western Sales dimension role would reference the Territory base dimension for its data.

A dimension role is a dimension column in a fact table that defines the foreign key that references a base dimension table. It always has an associated base dimension table. Dimension roles are defined within a constellation, and all fact tables in that constellation inherit the dimension role. Multiple roles within a single constellation can refer to the same base dimension.

The Momentum constellation has two special dimension roles: *group* and *indiv*, which are described below.

Demographic Base Dimensions and group and indiv roles (Momentum)

The demographic base dimension tables for individuals and groups are used by Momentum to generate lists of individuals and groups. A Momentum primary constellation must have one demographic base dimension table pointed at by the *group* dimension role. It can also have a base dimension table pointed at by the *indiv* dimension role. (These dimension roles cannot point to the same base dimension table.)

The base dimension table associated with the *group* dimension role must contain group demographic data, and the base dimension table associated with the *indiv* dimension role must contain individual demographic data; for example, account holders in a household, or employees in a company.

EpiCenter

An EpiCenter is a single logical Epiphany database installation that includes customer data in addition to control metadata. It physically consists of two databases: EpiMeta and its associated EpiMart.

EpiCenter Enterprise Manager (EpiCenter Manager)

EpiCenter Enterprise Manager is a Microsoft Windows program (EpiMgr.exe) for configuring EpiCenters. These key components of the Epiphany system are available via EpiCenter Manager: schema, ticksheet configuration, extraction, security metadata, the Adaptive Schema Generator, and metadata export/import.

EpiChannel

EpiChannel is the Epiphany program (**extract**) that executes extraction jobs. The person who runs EpiChannel enters parameters to connect to an EpiCenter and specifies jobs to be executed. EpiChannel then implements the extraction steps declared as metadata in the EpiCenter.

EpiMart

The physical database that contains actual customer data. The schema of tables in EpiMart is determined by the metadata in EpiMeta. The data contents of EpiMart tables are determined by the extraction steps in EpiMeta.

EpiMeta

EpiMeta is the metadata repository for an EpiCenter. It contains all control information for the EpiCenter and therefore defines the behavior of that EpiCenter. EpiMeta points to EpiMart via the special EpiMart data store, which is available in every EpiMeta.

Epiphany Report URL

The URL that is embedded in a mail message and invokes an Epiphany query.

External Column

An external column is a single column in an external table.

GLOSSARY

External Table

An external table is a table built in EpiMart, usually as a temporary table used during extraction. External tables must be declared as metadata (as opposed to being built outside of Epiphany) in order for the Adaptive Schema Generator to know of their existence. Otherwise, the table would be purged by the Purge EpiMart command.

Extraction Group

An extraction group is a set of extraction steps.

Extraction Step

An extraction step is a single atomic extraction operation. It can be either a SQL statement or a semantic Instance.

Extractor

An extractor is a list of extractor steps together with input and output data stores. It logically specifies a series of steps that move data from the input to the output data store.

Extractor Step

An extractor step is a single step of an extractor, which always points to an extraction group. In this way, extractors consist of an ordered list of extraction groups (which are ordered lists of extraction steps).

Fact Aggregate

A fact aggregate is a physical table in EpiMart that contains aggregated fact information for a single fact table.

Fact Clusters

Fact Clusters are tables MomentumBuilder builds to speed up runtime performance. When you associate a fact table in a Momentum Adjacent Constellation with a mini-dimension, the system creates copies of the fact table sorted by the mini-dimension.

Fact Column

A fact column is a single numeric column in a fact table.

Fact Counts

Fact counts are tables MomentumBuilder builds to speed up runtime performance. Associating fact tables in a Momentum Adjacent Constellation with a mini-dimension and a specific transaction type, such as Booked or Booked Returns causes statistics to be kept that help in selecting appropriate fact clusters.

Fact Table

A fact table is a physical table in EpiMart that contains numeric data in addition to references to dimension tables that specify attributes about the fact, such as who, what, when, and where the fact occurred.

Filter Block

On a ticksheet, a filter block controls the user's ability to filter on a single dimension column (for a dimension role of the ticksheet's constellation). The filter block also defines the appearance of the filter (for example, check box versus listbox).

Filter Element

A filter element is a single check box for filtering within a filter group, or a single entry within a listbox filter block.

Filter Group

A filter group is a logical grouping of filter elements within a filter block. It is applicable only to check box filter blocks.

Glossary Entry

A glossary entry provides end users with online definitions of the terminology used on a ticksheet, such as Units, Gross, and Sell-Through, which are hyperlinked to a glossary page. The creator of the ticksheet configures these entries via EpiCenter Manager or Web Builder.

GLOSSARY

Influence

Influence is one of the Relevance ticksheet types. Influence builds models of a company's data in order to identify the factors with the greatest impact on the business. For example, how do various attributes of a household influence whether or not a member purchases a product, and What influences whether a prospect responds to a direct mail offer?

Ind_Group_Joiner

This is a special fact table for Momentum that serves to connect the individual and group base dimension tables. It does not function in the same way as other fact tables in the Epiphany system. Although you cannot modify this dialog box, you do need to define how to extract data into the *Ind_Group_Joiner* table.

Job

A job is a top-level workflow object that defines a sequence of steps to be performed by EpiChannel. It also specifies the logging locations for that execution.

Job Step

A job step is a single step of a job, which can be either an extractor or a system call.

Leaf Dimensions

The dimension tables in a Momentum Adjacent constellation that point to non-demographic base dimensions.

Lists (Momentum)

Lists of demographic row keys that result from an end-user query, which reside permanently in EpiMart.

Measure

A measure is a single business calculation. It can be an arithmetic combination of measure terms using RPN (Reverse Polish Notation).

Measure Mapping

The mapping of ticksheet selection columns to measures.

Measure Sets

A measure set is used to define the target variable predicted by Influence. Measure sets are collections of other objects (measures and attributes). All measure sets include a Count measure, and most measure sets additionally include either a Sum measure or an attribute.

Measure Term

A measure term is a single component of a measure. It refers to the aggregation of a single fact column, such as *SUM(Order.net_price)*, with a particular transaction type. It can be combined with other measure terms to create a composite measure (business calculation).

Mini-Dimensions

A *mini-dimension* is a subset of a base dimension table used for Momentum queries.

Momentum

Momentum allows you to create lists of demographic dimensions, such as Customers, Resellers, Sites, Households, Individuals, Contacts or other similar entities. A list is always generated by using the Momentum user interface to apply successive filtering. Once you have defined a filter in Momentum, you can find out how many matches it corresponds to in the database, either exactly or approximately.

Momentum filters can be for individual data only, or for both individual and group, or household data. You can filter on attribute data and, optionally, on transactions. A transaction is usually an action such as bought, returned, called call center, received promotion or responded to campaign.

GLOSSARY

Momentum Adjacent Constellation

The Momentum Adjacent Constellations allow you to generate lists based on behavioral or transactional aspects of your customers. Momentum adjacent constellations must have at least one dimension role pointing at a Momentum demographic base dimension table. See *Demographic Base Dimensions and group and indiv roles*.

MomentumBuilder

MomentumBuilder is the executable that creates special Momentum tables, such as mini-dimension tables and clusters.

Momentum Constellation

The Momentum Constellation is the special Momentum constellation that has the Momentum fact table *Ind_Group_Joiner*, and the group dimension role. The Ticksheets folder for configuring Momentum ticksheets is part of this constellation. It allows you to generate lists based on demographic data about your customers.

Pull/Push SQL Statement

A pull/push SQL statement is a type of extraction step that issues SQL against an input data store and then pushes the result set into a table in the output data store of an extractor.

Queryable Fact Column

A fact column upon which a user can query transactions using Momentum.

Query Machinery

The component of the Epiphany Application Server that communicates with EpiMart and actually issues SQL statements against the DBMS. The query machinery is a common component of Epiphany's front-end applications.

Relevance

Relevance ticksheets enable users to analyze a company's data in order to find trends, project quarterly results, profile and segment customers, and find correlations and anomalies in the company's data. Also see *Influence*.

Report

A report is a saved set of ticksheet settings from a previous query that can be viewed again. Users and groups are granted permission to use or modify Reports. Users who request the same Report (and have the same dimension column access rights) will receive the same output (reports, charts, and so forth).

Report Gallery

The Report Gallery is a component of Epiphany's front-end user interface that allows users to view reports. EpiCenter Manager also contains a Report Galley in the Security folder that has additional features for administrative use.

Scrutiny

Scrutiny is a debugging tool available through EpiCenter Manager that you can run periodically, or whenever you suspect problems with the EpiMeta or EpiMart tables, or if the Application Server fails to start. Scrutiny loops through a series of tests to determine if any problems exist. If it finds one, it explains the problem, asks if you want to correct it, and then corrects it, if possible.

Security Manager

Security Manager (**SecMgr.exe**) is an Epiphany program for assigning access rights to log onto the system, as well as permission to view, modify, and save ticksheets. The Security Manager program, which is a subset of EpiCenter Manager, allows its users to access the security features of EpiCenter Manager, without having access to its full-functionality.

Selection Column

A selection column is a single column in the measure section of a ticksheet. The combination of one value from each selection column translates into a single measure in the report output.

GLOSSARY

Selection Column Element

A selection column element is a single value in a selection column. For instance, a selection column might contain the elements *Gross*, *Net*, and *Return*.

Semantic Instance

A semantic instance is a post-compilation SQL program. When a semantic template is applied to either a base dimension or fact table, the template becomes instantiated as an actual SQL program that can be used to accomplish specific business rules during extraction.

Semantic Template

A semantic template is a generic SQL program intended to accomplish specific business rules (these rules are referred to as semantic types) during extraction. A semantic template does not refer to actual column or table names. Only when the template is applied to a base dimension or fact table (via a semantic instance) does the SQL contained within it refer to real column and table names.

Semantic Type

A semantic type refers to the logical business process for which the semantic template is applied.

Source System

A source system is the logical notion of a source of business data. Two physical databases (one a backup of the other) might represent the same source system within Epiphany.

SQL Statement

A SQL statement is an extraction step that issues custom SQL against an input data store. The results of the SQL statement can either be discarded or used to push data into a table in the output data store.

Stand-Alone SQL Statement

A stand-alone SQL statement is a SQL statement whose results are discarded.

System Call

A system call is an extraction step that causes an operating system program to be invoked.

Ticksheet

A ticksheet is a form that end users open in a Web browser and use to submit queries to the data warehouse. It is called a ticksheet because users make selections by ticking (selecting) items.

The ticksheet developer uses the Ticksheet dialog box in EpiCenter Manager or Web Builder to construct these ticksheets.

Transaction Filters

Transaction filters allow Momentum ticksheet users to filter dimension demographics by behavior, such as “Show me how many people *called* (the behavior) Customer Support from Region Y in 1998.” Transaction filters are always associated with a fact table and transaction type, and a set of regular attribute filters. Optionally, transaction filters can be configured to apply a measure filter to the result of that filter set. See *Momentum*.

Transaction Type

Transaction types distinguish rows with different interpretations in the same fact table. For example, an Order fact table might contain both a BOOK and SHIP transaction type, differentiated by the value of the column *transtype_key*.

Measure terms created via EpiCenter Manager or Web Builder specify a transaction type in addition to a fact column. This allows the summation of only those rows in a fact table with the same transaction type.

GLOSSARY

Web Builder

Web Builder (**WebBuilder.exe**) is an Epiphany program for configuring user-interface metadata, including measures, ticksheets, and glossary entries. The Web Builder program, which is a subset of EpiCenter Manager, allows its users to configure ticksheets, without having access to the full-functionality of EpiCenter Manager.

09625518 072500

INDEX

A

- A and B tables 50, 83
 - browsing aggregates 105, 113
 - most current 339
 - toggling 50, 221
- Access rights
 - group/user precedence 201
 - user/group precedence 205
- Acrobat PDF Files 16
- Actual tables 36, 103, 345
- Actuals metadata, Export All 326
- Adaptive Schema Generator 41, 305, 326, 345
- Add Column dialog box 207
- Add Job Steps dialog box 189
- ADD_DAYS SQL macro 270
- ADD_MONTHS SQL macro 270
- Administrative rights 202
- Administrator group 256
- Agg metadata 326
- AGG system call 258
- agg.exe. *See* Aggbuilder
- agg_timestamp, Aggbuilder log directory 58
- Aggbuilder 30, 35, 42, 54, 58, 102, 326, 345
 - browsing dimension 105
 - browsing fact 113
 - log file structure 58
 - program steps 59
- Aggregate building
 - extraction phase 42
 - invoking 58
- Aggregate Group dialog box 105, 115, 116
- Aggregate groups 115, 345
 - and dimension roles 116
 - default group 117
- Aggregate navigation 232, 346
- Aggregate operators 110
- Aggregate tables 62
- Aggregates
 - base dimension 347
 - Best & Worst 151
 - browsing dimension 105
 - browsing fact 113
 - building for Influence 160
 - fact 55
 - Influence 151
 - Lifecycles 151
 - optimal number of 105
 - Profiling 150
 - purpose of 55
 - Quarter Projections 150
 - Relevance 150
 - Trends 151
 - vs. custom fact indexes 111
- Aggregation 62
- AGGVERIFY system call 259
- all_groups_, MomentumBuilder table 62
- all_individuals_, MomentumBuilder table 62
- Allow Anonymous authentication 254
- API, native for EpiCenter 49
- Application Log Full Error 342
- Application Server 42, 346
 - aggregate navigation 232
 - aggregates at query time 59
 - aggregates rebuilt 233
 - APPSERVER.LOG 343
 - cannot log in 331–333
 - config_master table 232
 - EpiAppService NT Service 237
 - error messages 330–343
 - GIFs 338
 - log files 228, 245
 - log naming conventions 246
 - memory for 334
 - mx64M 334

INDEX

- proxy logging 243
- query machinery 356
- refresh time 235
- RefreshApp 236
- refreshing 232, 233, 234
- Registry keys 240
- restarting 233
- running from command line 231
- security 250
- security logs 333
- Security Manager log 248
- Service Control Manager 333
- setting up new domain 255
- SRV log 343
- starting from console 230
- starting/stopping Windows NT service 227
- time navigation 233
- verifying operation 228
- Windows NT authentication 231
- Windows Registry 232
- ApplicationServerPort Registry keys 240
- APPSEVER.LOG 333, 343
- AppServerHost Registry key 240
- APPSEVERHOST system call 259
- AppServerLogVerbosity Registry key 241
- APPSEVERPORT system call 259
- AppSessionTimeout Registry Key 241
- Arithmetic operators, RPN 121
- ASSERT_INDEX_EXISTS SQL macro 270
- Attribute dialog box 131, 133
- Attribute roles 346
 - assigning on ticketsheets 132
 - Relevance ticketsheets 146
- Attributes 25, 346
 - Clarity 130
 - modifying 133
 - Momentum 171
 - Relevance 130
 - steps for defining 133
 - ticketsheet 130
- Attributes dialog box 133
- Authentication
 - basic 254
 - modules 252
 - Pass through 253
 - Windows NT domain 252

B

- Backlog types 121, 347
- Base dimension aggregates 347
- Base Dimension dialog box 98, 99, 102, 105, 107, 114, 161, 168
 - selecting physical types 299
- Base dimension staging tables
 - populating 307
 - queries with joins 306
 - SQL statements 303
- Base dimension tables 23, 56, 98, 347
 - defining 99
- BASE HREF tag 338
- Base tables 56, 347
- Basic authentication 254
- BEGIN_ASSERT_INDEX SQL macro 270
- Best & Worst
 - aggregates 151
 - default ticksheet 147
- Bookings 319
- BOOL_TO_YN SQL macro 270
- Breakpoints, setting 79
- Build Registry key 241

C

- Calendars
 - 4-4-5 quarters 32, 217
 - fiscal quarter 32
- Can be used in Momentum filter 101
- CASE_BEGIN SQL macro 271
- CASE_ELSE SQL macro 271
- CASE_ELSEIF SQL macro 271
- CASE_END SQL macro 271
- CAT SQL macro 271
- CBIN_VAL SQL macro 272
- CHAR_1 SQL macro 272
- CHARTSLOGFILE system call 259
- ChartsOutputDir Registry Key 241
- CHARTSOUTPUTDIR system call 259
- chn_timestamp, EpiChannel log directory 58

INDEX

- Choose EpiCenter dialog box 88, 92
 - Choose Groups dialog box 130, 206
 - Choose Measure dialog box 156
 - Choose User dialog box 130
 - Clarity 13, 20, 31, 124
 - Tables & Charts 124
 - timesheet configuration 126–146
 - Classification targets 158
 - Classification tree, Influence 152
 - Cleansing tools, used by external tables 50
 - Clustered index, fact table 111
 - Clusters 170
 - guidelines for 170
 - setting up on fact table 173
 - Column sets 102, 116
 - COLUMN_FILTER SQL macro 266
 - COLUMN_LAST_VALUE SQL macro 266
 - COLUMN_RANGE_FILTER SQL macro 266
 - com.epiphany.security.EpiNTLogon security module 250
 - com.epiphany.security.SecurityManager class 246
 - com.epiphany.server.Server class 245
 - Commands
 - epiappservice 237–240
 - extract.exe 186
 - jre 230, 334
 - MomentumBuilder.exe 60
 - refresh.exe 232–237
 - RefreshApp.exe 232, 236
 - Regedit.exe 329
 - SecMgr.exe 357
 - WebBuilder.exe 360
 - config_master table 50, 232, 285
 - Configuration dialog box 97, 119, 126, 162, 163, 216, 285–294, 339
 - Timesheet Types 293
 - Configuration, EpiCenter 97
 - Constellations 21, 348
 - multiple 86
 - setting up 106–176
 - Copy Timesheet Items 145
 - COUNT DISTINCT, SQL operator 120
 - Count feature, Momentum 163
 - Count Unjoined semantic type 52, 316
 - COUNT, SQL operator 120
 - COUNT_ROWS SQL macro 272
 - COUNTER SQL macro 272
 - Counts 170, 174
 - CREATE_INDEX_IF_NOT_EXISTS SQL macro 272
 - CURR SQL macro 265
 - Currencies
 - multi-currency option 289
 - system usage 289
 - Currency measurement 120
 - current_datamart A / B 286
 - current_datamart value 50, 233, 338
 - current_datamart, EpiCenter Manager option 83
 - Custom Fact Index semantic type 52, 63, 111, 317, 348
 - Custom fact indexes 62–64, 111, 348
-
- D**

 - Data is valid as of... 287
 - Data merging extraction phase 42, 51
 - Data Store dialog box 177, 179, 181
 - Data stores 42, 43, 348
 - creating 177
 - EpiMart 43, 180
 - generic ODBC 179
 - JobFileLog 43, 180
 - LoggingDB 44, 180
 - Oracle 179, 262
 - SQL Server 179
 - Data warehouses 15, 19, 20, 44
 - Database administration tools 40
 - Database server
 - registering 87–89
 - DATABASE system call 259
 - DatabaseName Registry key 241
 - DatabasePassword Registry key 241
 - DatabaseServer Registry key 241
 - DatabaseType Registry key 241
 - DatabaseUsername Registry key 241
 - Dataset dialog box 127

INDEX

- Datasets 128, 213, 348
 - assigning ticketsheets to 127
 - ordering of 214
- Date base dimension tables 98
- Date Dimension dialog box 91, 117
- Date dimension fields 295
- Date dimension tables 24, 31, 117, 215, 349
 - populating 91, 214, 216
- Date range, EpiMart 216
- Date, displayed on reports 339
- Date.fy_name 205, 209
- DATE_FILTER SQL macro 267
- date_key 214, 309, 349
- DATE_LAST_VALUE SQL macro 267
- date_modified column 305
- DATE_RANGE_FILTER SQL macro 267
- date_type 286
- DBNOW SQL macro 273
- DBVENDOR system call 259
- DDL_BEGIN SQL macro 273
- DDL_BEGIN_NO_DECLARE SQL macro 273
- DDL_END SQL macro 274
- DDL_EXEC SQL macro 274
- Debug level, job 191
- DEBUG SQL macro 284
- DEBUG_LEVEL system call 259
- Debugging. *See* Scrutiny
- DECLARE_BEGIN SQL macro 275
- DECLARE_BODY SQL macro 275
- Defaults report folder
 - user and group 210
- Degenerate Dimension dialog box 108
- Degenerate dimensions 108, 348, 349
- Description Registry key 242
- Dialog boxes
 - Add Column 207
 - Add Job Steps 189
 - Aggregate Group 105, 115, 116
 - Attribute 131, 133
 - Attributes 133
 - Base Dimension 98, 99, 102, 105, 107, 114, 161, 168, 299
 - Choose EpiCenter 88, 92
 - Choose Groups 130, 206
 - Choose Measure 156
 - Choose User 130
 - Configuration 97, 119, 126, 162, 163, 216, 285–294, 339
 - Data Store 177, 179, 181
 - Datasets 127
 - Date Dimension 91, 117
 - Degenerate Dimension 108
 - Dimension Column 101, 172
 - Dimension Role 107, 162
 - Execute Job 191
 - Exporting Metadata 218
 - External Tables 299
 - Extractor 177, 178, 181
 - Extractor Steps 181, 182, 184, 188
 - Fact Column 110
 - Fact Table 108, 110, 111, 112, 113, 173
 - Filter 138, 176
 - Filter Elements 138
 - Filter Group 137
 - Filters 135
 - Find Report 211
 - Generate Schema 214
 - Glossary Entry 134, 142
 - Importing Metadata 219
 - Influence ticketsheet 155
 - Initialize EpiCenter 89, 90, 161
 - Job 189, 190, 191
 - Job Steps 192
 - Measure 118, 120, 122, 140, 157, 175
 - Measure Builder 118, 122, 123
 - Measure Column Element 141
 - Measure Sets 156
 - Measures 122
 - Momentum ticketsheet 171
 - New Constellations 161
 - Populates fact table option 307
 - Register EpiCenter 89
 - Report 95, 144, 212
 - Semantic Instance 188
 - Server Properties 88, 221, 222
 - SQL Query 139
 - SQL Statement 185, 186, 187, 304
 - SQL Statement, Tables References option 186, 304
 - SQL Statement, Template feature 187
 - Ticketsheet 126, 129, 134, 141, 144, 145, 146
 - Transaction Filter 174, 175
 - Truncate Tables before Extraction 199
 - User 205

INDEX

- Dimension aggregates 59, 102, 105
 - defining 103
 - trade-offs 105
 - Dimension Column dialog box 101, 172
 - Dimension column sets 102
 - defining 103
 - multiple 105
 - Dimension columns 349
 - displaying values of 208
 - restricting access to groups 203
 - restricting access to users 207
 - UNKNOWN 102
 - Dimension Role dialog box 107, 162
 - Dimension roles 23, 348
 - custom fact index 112
 - defining 107
 - in aggregate group 116
 - Dimension semantic types 52, 311–316
 - Dimension tables 20
 - defining dimension roles 107
 - hierarchical relationship of 25
 - number required 99
 - Dimensions
 - attributes of 25
 - degenerate 108
 - DimRoleName__sskey 309
 - DIRNAME system call 259
 - Domains 254
 - Driver Registry key 243
 - DROP_INDEX SQL macro 275
 - DROP_TABLE_IF_EXISTS SQL macro 275
 - DSN Registry key 242
 - DSN system call 260
 - Duplication feature, EpiCenter Manager 94
- E**
- Elements, measure column 118
 - ELSE SQL macro 275
 - E-mail
 - configuring for job status 196
 - EpiCenter Manager default address 198, 285
 - password 285
 - End of Extraction 50, 58, 183
 - End of Extraction job 339
 - END_ASSERT_INDEX SQL macro 275
 - END_IF SQL macro 276
 - end_year 286
 - EOS SQL macro 276
 - EpiAdminServer 248
 - epiappservice command 237–240
 - EpiAppService. *See also* epiappservice 237
 - EPIBIN system call macro 54, 61, 260
 - EpiCenter 13, 21, 44
 - current configuration 97
 - initializing 89–92
 - EpiCenter Manager 30, 39, 86, 94, 233, 234, 353
 - cloning tables 94
 - configuration 95
 - conventions 94
 - defining base dimension tables 99
 - duplication feature 94
 - Export all command 326
 - Generate Schema 214
 - invoking commands 93
 - measure terms 359
 - registering server 87–89
 - starting the program 87
 - updating data 94
 - window 93
 - EpiCenter Manager Report 95
 - EpiCenter Manager tables
 - _A and _B sets 106
 - EpiCenters 39
 - appropriate number of 86
 - backing up definitions 323
 - producing new 219
 - registering existing 92
 - setting up 97
 - unregistering 94
 - EpiChannel 30, 40, 41, 43, 45, 54, 58, 257
 - Also see* extract.exe
 - error messages 78
 - logging 81
 - memory 48
 - output 75, 80
 - selecting debug level 186
 - trial-run model 79
 - verbose debugging option 78
 - verbosity levels 191

09625518, 1072500

INDEX

- EpiChannel log file 314, 315, 317
 - sskey rows 312, 316
- EpiChannel output 75
- EPIINT SQL macro 276
- EPIKEY SQL macro 276
- EpiLoginException violations 230
- EpiMart 21, 30, 42
 - base tables 56
 - data range 216
 - data store 43, 180
 - initial load of 315
 - schema changes in 35
 - tables 39, 83
- EpiMart dimension column values displaying 208
- EpiMart tables 30
 - building 35
 - mirroring of 106
 - naming conventions 35
 - purging 199
 - testing with Scrutiny 223
 - toggling A and B 50
- EpiMeta 21, 28, 29, 35, 39, 323
 - cloning 326
 - database table schema 85
 - tables 29, 324
- EpiNTLogon 255
 - Windows NT authentication module 250
- EpiPassThruLogon module 333
- Epiphany
 - database schema 19
 - Release 3.4 features 14
 - setting up system users 205
 - system overview 27–29
- Epiphany Application Server. *See* Application Server
- Epiphany Application Suite 31
- Epiphany Customer Support 329
- Epiphany macros
 - SQL 262–284
 - system call macros 257–261
 - translation of 265
- Epiphany software authentication modules 252
- Epiphany vendor-independent macros 264
- Epiphany.dll proxy 225, 243
- Error messages
 - Application Log Full 342
 - Application Server 330–343
 - Cannot Connect to the Server 330
 - EpiChannel 78
 - Object Not Found 340
 - out of memory 334
 - Refresh program fails 342
 - Registry Editor warning 329
 - SQL Server 330
 - User Cannot Log In 331
 - VirtualMartDatabase Key Missing 334
- EXC system call 260
- EXC_ARGS system call 260
- EXC_CMD system call 260
- EXCVERIFY system call 260
- Execute Job dialog box 191
- Export All command 326
- Export machinery 324
- Export tables
 - Export_col 327
 - Export_rel 327
 - Export_row 327
 - Export_status 327
 - Export_tbl 327
 - Rel_parent 327
- export.mdb 219
- Export_col table 327
- Export_rel table 327
- Export_row table 327
- Export_status table 327
- Export_tbl table 327
- Exporting Metadata dialog box 218
- Exporting ticksheets 324
- External tables 41, 50, 194
 - cleansing tools 50
 - input to staging queries 310
 - last_extract_date 50, 194
 - truncation of 198
- External Tables dialog box
 - selecting physical types 299
- extract.exe
 - Execute button in EpiCenter Manager 192
- extract.exe command 186
 - Also see* EpiChannel
 - directory location 54

Extraction
 aggregate building phases 42
 data merging phase 42, 51
 defining groups 184
 extraction steps for groups 184
 filter macros 182
 groups 182
 load phase 41, 51
 multi-stage 53
 setting up for EpiCenter 177
 stages 41
 steps 184
 Extraction Date Unknown 339
 Extraction groups 46
 Extraction statements 50
 Extraction step 46
 Extractor dialog box 177, 178, 181
 Extractor filters 180
 Extractor steps 46
 Extractor Steps dialog box 181, 182, 184, 188
 Extractors 41, 46, 47
 adding as job step 193
 defining 181
 End of Extraction 50, 58
 Extractor Steps dialog box 182
 shared by jobs 47

F

Fact aggregates 55, 59
 browsing 113
 Fact Column dialog box 110
 Fact columns
 defining 110
 queyable 356
 Queyable option 110
 Fact semantic types 52, 316–322
 Fact staging tables 64
 loading 322
 Fact Table dialog box 108, 110, 111, 112, 113, 173
 Fact tables 20, 25, 34
 clustered index 111
 defining 108–109
 reloading populated 321

FACTMONEY 276
 FACTMONEY SQL macro 276
 FACTQTY SQL macro 276
 Facts
 in EpiMart 25
 rows with zero facts 311
 SQL limits 84
 FILENAME system call 260
 Filter dialog box 138, 176
 Filter elements 136
 defining 138–140
 Filter Elements dialog box 138
 Filter Group dialog box 137
 Filter groups 136
 defining 137–138, 138–140
 Filter Households ticksheet type 171, 293
 Filter Individual Momentum ticksheet 171
 Filter Individual ticksheet type 171, 293
 filter_block_key column 323
 FilterGroupCols, Momentum role type 171
 FilterIndividualCols, Momentum role type 171
 Filters
 defining 134–140
 Momentum ticksheet 172
 ticksheet 134
 Filters dialog box 135
 Find Report dialog box 211
 First Dimension Value semantic type 314
 Flat files, source system 43
 FLOAT SQL macro 276
 Foreign keys 24, 31
 to date dimension 31

G

Generate Schema command 35, 36, 214
 Generate Schema dialog box 214
 GIFs, generated by Application Server 338
 Glossary entries 213
 adding filter group 138
 ticksheet items 134
 Glossary Entry dialog box 134, 142

INDEX

Group by flags. selecting 104
 GROUP BY statements 306
 GROUP BY, SQL clause 104
 Groups
 assigning ticksheet access to 129
 setting up 201–205

H

Household, Momentum ticksheet type 125
 HOUSEHOLD_CARDINALITY 291

I

IDENTITY SQL macro 276
 IF SQL macro 276
 IIS security 254
 IIS Web server. *See* Web server
 IJ_FROM SQL macros 277
 ikey 318
 Import machinery 324, 325
 Importing Metadata dialog box 219
 Ind_Group_Joiner fact table 164
 index.pdx, Acrobat full-text index 16
 Indexed for fast Clarity and Relevance search option 101
 Indexing 62
 custom fact 111
 fact table clustered index 111
 leading term 111
 Individual, Momentum ticksheet type 125
 INDIVIDUAL_CARDINALITY 291
 Influence 125
 aggregates 151, 160
 classification tree 152
 determining primary dimension 154
 determining source attributes 154
 determining the target 154
 measure sets 154, 156
 Momentum lists as attributes 159

 regression tree 152, 153
 tips 159
 Influence ticksheet dialog box 155
 Influence ticksheets
 configuring 152
 primary dimension 153
 setting up 153–158
 source attributes 153
 target 153
 Initial Load Dimension semantic type 52, 313, 315
 Initial Load Fact semantic type 52, 321
 INITIAL_LOAD SQL macro 269
 Initialize EpiCenter dialog box 89, 90, 161
 INSTANCE_NAME system call 260
 InstanceRootDir Registry key 242
 INSTANCEROOTDIR system call 260
 INSTR SQL macro 277
 Inventory data. transactional system 34
 ISQL 339
 iss sskey 312
 ISS system call 260

J

Job data store roles 54
 Job dialog box 189, 190, 191
 Job steps 46
 ordering 189
 Job Steps dialog box 192
 JOB_NAME system call 261
 JobFileLog data store 43, 180
 JobLogFile 179
 Jobs 75
 debug level for 191
 default 189
 disabling 190
 e-mail notice of status 196
 monitoring 81–82
 role of 45
 JOIN_LEFT_OUTER SQL macro 277
 JOIN_RIGHT_OUTER SQL macro 277
 JOIN_WHERE SQL macro 278
 jre command 230, 334

L

last_extract_date 50, 194, 287, 339
 Latest Dimension Value semantic type 313
 Leading terms, fact table indexing 111, 348
 Leaf dimensions 165, 167
 LENGTH SQL macro 278
 Lifecycles 125
 aggregates 151
 default ticksheet 149
 Listboxes
 dynamic 136
 filter block 136
 Load extraction phase 41, 51
 Log database 58, 81
 Log device 81
 Logging Data Store 179
 Logging, proxy 243
 logging_mssql.sql script 179
 LoggingDB data store 44, 180
 Logs
 location of 80
 role of 80
 LOJ_FROM SQL macro 278

M

Macros
 Epiphany vendor-independent 264
 semantic instance 46
 SQL 49
 Mail Profile name 198, 285
 MARTDBNAME SQL macro 269
 MAX_SYS_DATE SQL macro 278
 Measure Builder dialog box 118, 122, 123
 Measure Column Element dialog box 141
 Measure column elements 118
 Measure dialog box 118, 120, 122, 140, 157, 175
 Measure mapping
 defined 118
 verifying 143
 Measure sets
 Influence 154, 156
 Measure Sets dialog box 156
 Measure terms 120, 359
 Measure units 119, 289
 defining 289
 symbols for 289
 Measurement units 120
 Measures 118, 348
 defining for a constellation 118
 mapping elements to 143
 Measures dialog box 122
 Metadata 21
 current state of 215
 exporting individual objects 218
 exporting/importing of 217
 overwriting 325
 primary keys in tables 323
 ticksheet 324
 METADBNAME SQL macro 269
 Microsoft Access
 database 218, 221
 database tables 324
 Export database 326
 Microsoft Internet Explorer 342
 Microsoft Outlook Exchange
 EpiChannel e-mail 197
 Microsoft SQL Server
 cannot connect to 330
 error messages 330
 Physical type values 299
 TCP/IP sockets 336
 Microsoft SQL Server data stores 179
 min_sample_invlog10 286
 Mini-dimensions 168–169
 MODULO SQL macro 278
 mom_inlist_limit 287
 Momentum 14, 20, 31, 125, 160, 291
 Can be used in filter option 101
 clusters 170, 173
 constellations 161, 165
 count feature 163
 counts 170, 174
 defining transaction filters 174–176
 Filter Household ticksheet type 171
 Filter Households ticksheet type 293

INDEX

- Filter Individual 171
- Filter Individual ticksheet type 293
- FilterGroupCols role type 171
- FilterIndividualCols role type 171
- HOUSEHOLD_CARDINALITY 291
- INDIVIDUAL_CARDINALITY 291
- leaf dimensions 167
- mini-dimensions 168–169
- modifying default labels 291
- MOMENTUM_HOUSEHOLD_LABELS 291
- MOMENTUM_INDIVIDUAL_LABELS 292
- MOMENTUM_MEASURE_MODE 292
- Option labels for ticksheets 162
- order of extraction for 60
- queryable fact column 356
- saving filter settings 176
- setting up attributes 171
- slowly changing dimensions 165
- testing with Scrutiny 223
- ticksheet filters 172
- transaction filters 169
- Momentum Adjacent constellations 86, 166, 348
 - setting up 166
- Momentum constellations 348
- Momentum lists
 - tips for using with Influence 159
 - using as Influence attributes 159
- Momentum sampling tables 214, 215
- Momentum Ticksheet dialog box 171
- Momentum ticksheet types 125
- MOMENTUM_HOUSEHOLD_LABELS 291
- MOMENTUM_INDIVIDUAL_LABELS 292
- MOMENTUM_MEASURE_MODE 292
- MomentumBuilder 60
 - all_groups_table 62
 - all_individuals_table 62
 - default job 61
 - Java file for 61
 - system call 189
 - system call job step 195
 - verifying extraction 62
- MomentumBuilder.exe 60
 - command-line arguments 60–61
- Monitors 190
- MSSQL Server SQL Service 233
- Multi-currency option 289
- mx64M parameter, Application Server 334

N

- NBIN_VAL SQL macro 279
- Netscape Navigator 342
- New Constellations dialog box 161
- NEXT SQL macro 265
- No Results Available for a Query 338
- NO_FROM_LIST SQL macro 279
- NOT_DEBUG SQL macro 284
- NOT_INITIAL_LOAD SQL macro 269
- NT. *See* Windows NT
- NTLM authentication 254
- NUMBER() SQL macros 279
- number_of_years 286
- NVL SQL macro 279

O

- Object Not Found 340
- ODBC
 - data store 179
 - layer 43
 - source system 43
 - SQL driver 330
- fiscal_year start_m 286
- Option labels
 - modifying default Momentum 291
 - ticksheet 290
- Options Labels
 - Momentum ticksheet 162
- Oracle
 - data store 179
 - Physical type values 299
 - SQL macros 262, 279
 - tablespaces 262, 263
- ORACLE SQL macro 279
- Out of Memory exception error 334
- Output Data Store 178

P

Pass-through authentication 253
 PASSWORD system call 261
 PATH system call 261
 PDF files 16
 full-text index 16
 Per cent measurement 120
 permission 249
 Permissions
 default for new users 205
 effect of changing on Application Server 234
 Security Manager 222
 setting for ,viewing folder/report 212
 timesheet 249
 Physical type values
 Oracle 299
 SQL Server 299
 Physical types 299
 selecting for fact column 110
 Pipelined semantic type 320
 Populates a dimension option 304
 Populates fact table option 307
 Postfix 289
 Preserve existing items option 145
 Primary dimension
 determining 154
 Influence timesheet 153
 Primary keys 31, 323
 correspondence to sskeys 305
 process_key 309, 318, 319, 320, 321
 Profiling
 aggregates for 150
 default timesheet 147
 Relevance Profiling 125
 PROGRAM_NAME system call 261
 Propagate button 288
 proxy.log file 242
 ProxyLogFile Registry key 242
 Public Defaults folder 210
 Pull/push SQL statements 356

Q

QM_randomly_generated_alphabetic_sequence_use
 name 333
 Quarter Projections 125
 aggregates 150
 default timesheet 149
 Query machinery 42, 233, 356
 Query performance, improving 62
 Query run time
 setting for user 206
 Queryable fact column. Momentum 356
 Queryable option, fact column 110

R

RAISE_EXCEPTION SQL macro 280
 RDBMS 27
 indexes 32
 physical types for platform 299
 server 43
 Refresh system call 189
 refresh.exe command 232–237, 334
 fails 342
 RefreshApp.exe command 232, 236
 Regedit.exe command 329
 Register EpiCenter dialog box 89
 Registry Editor warning 329
 Registry keys
 Application Server 240
 ApplicationServerPort 240
 AppServerHost 240
 AppServerLogVerbosity 241
 AppSessionTimeout 241
 Build 241
 ChartsOutputDir 241
 DatabaseName 241
 DatabasePassword 241
 DatabaseServer 241
 DatabaseType 241
 DatabaseUsername 241
 Description 242
 Driver 243
 DSN 242

INDEX

InstanceRootDir 242
ProxyLogFile 242
SecurityClass 243, 250, 253
SystemLogDir 242
SystemLogDirWebpath 242
SystemLogWebDir 249
TempDirGarbageLifetime 242
TemplateDir 242
REGISTRY_EPIPATH system call 261
REGISTRY_ROOT system call 261
Regression targets 158
Regression tree, Influence 152, 153
Rel_parent table 327
Relational database management source systems. *See*
RDBMS
Relevance 13, 20, 31, 124
 aggregates for 150
 Best & Worst 124, 147
 date dimension for trends 216
 default ticksheets 147
 Influence 125
 Lifecycles 125
 Profiling 147
 Quarter Projections 125, 149
 ticksheet attribute roles 146
 ticksheet configuration 126–158
 Trends 125, 148
Relevance ticksheets 356
Reload Max Date semantic type 322
RENAME_OBJECT SQL macro 280
Report dialog box 95, 144, 212
Report Gallery 144, 200, 209, 357
Reports 357
 copying/moving 211
 date displayed on 339
 folders for saved 211
 saved for ticksheet 144
Result Page Error
 Extraction Date Unknown 339
Reverse Polish Notation 120, 122–124
 arithmetic operators 121
 translation of 122
RPN. *See* Reverse Polish Notation

S

SAM 254, 255
Sampling tables, Momentum 214, 215
Save and Restore Manager 249
SAVE_RESTORE 333
Schema 345
 changes in Epimart 35
 definitions 36
 generating 214
 operations 37
Scrutiny debugging tool 223, 357
 testing Application Server 227
SecMgr.exe 222
SecMgr.exe command 222, 357
 See also Security Manager
Security
 access rights 200
 authentication 200
 Epiphany system 200
 Web browser caching 199
 Windows NT groups 200
Security Access Monitor. *See* SAM
Security logs
 Application server 333
 APPSERVER.LOG 333
 QM_randomly_generated_alphabetic_sequence...
 333
 SAVE_RESTORE 333
Security Manager 87, 222, 234, 250, 357
SecurityClass Registry key 243, 250, 253
SecurityManager log 246, 248
SELECT DISTINCT query 307
SELECT statements 307
SELECT INTO BEGIN SQL macro 280
SELECT INTO BODY SQL macro 280
Selection column 357
Semantic Instance dialog box 188
Semantic instances 35, 40, 42, 46, 49
 defined 51
 setting up 188–189
Semantic templates 305
 defined 51
Semantic transformation 51

096255318 072500

INDEX

- Semantic types
 - Count Unjoined 52, 316
 - Custom Fact Index 52, 63, 111, 317, 348
 - defined 51
 - defining 52
 - dimension 52
 - fact 52
 - First Dimension Value 314
 - Initial Load Dimension 52, 313, 315
 - Initial Load Fact 52, 321
 - Latest Dimension Value 313
 - Pipelined 320
 - Reload Max Date 322
 - Slowly Changing Dimensions 311, 315
 - Transactional 317
 - transactional 317
 - Transactional/State-like 318, 322
 - Transactional/State-like/Force Close 319
- Semantic types base dimension and fact tables 52
- Semantics 39, 40
- Server Properties dialog box 88, 221, 222
- SERVER system call 261
- Servers
 - unregistering 94
- Service Control Manager
 - fails to start 333
- Shipments 319
- Slowly Changing Dimensions 315
- Slowly changing dimensions 313
 - Influence 160
 - Momentum 165
 - semantic type 311
- Slowly changing dimensions semantic type 315
- SMALLDATE SQL macro 280
- SMALLINT SQL macro 280
- Source attributes
 - Influence tick sheets 153, 154
- Source system identifier keys. *See* sskeys
- SQL
 - dialect problems 186
 - limits for facts 84
 - sample for populating a table 187
- SQL macros 49, 257, 268, 276, 305
 - ADD_DAYS 270
 - ADD_MONTHS 270
 - ASSERT_INDEX_EXISTS 270
 - BEGIN_ASSERT_INDEX 270
 - BOOL_TO_YN 270
 - CASE_BEGIN 271
 - CASE_ELSE 271
 - CASE_ELSEIF 271
 - CASE_END 271
 - CAT 271
 - CBIN_VAL 272
 - CHAR_1 272
 - COLUMN_FILTER 266
 - COLUMN_LAST_VALUE 266
 - COLUMN_RANGE_FILTER 266
 - COUNT_ROWS 272
 - COUNTER 272
 - CREATE_INDEX_IF_NOT_EXISTS 272
 - CURR 265
 - DATE_FILTER 267
 - DATE_LAST_VALUE 267
 - DATE_RANGE_FILTER 267
 - DBNOW 273
 - DDL_BEGIN 273
 - DDL_BEGIN_NO_DECLARE 273
 - DDL_END 274
 - DDL_EXEC 274
 - DEBUG 284
 - DECLARE_BEGIN 275
 - DECLARE_BODY 275
 - DROP_INDEX 275
 - DROP_TABLE_IF_EXISTS 275
 - ELSE 275
 - END_ASSERT_INDEX 275
 - END_IF 276
 - EOS 276
 - EPIINT 276
 - EPIKEY 276
 - Epiphany 262–284
 - FACTQTY 276
 - FLOAT 276
 - IDENTITY 276
 - IF 276
 - IJ_FROM 277
 - INITIAL_LOAD 269
 - INSTR 277
 - JOIN_LEFT_OUTER 277
 - JOIN_RIGHT_OUTER 277
 - JOIN_WHERE 278
 - LENGTH 278
 - LOJ_FROM 278
 - MARTDBNAME 269
 - MAX_SYS_DATE 278
 - METADBNAME 269

09625318 072500

INDEX

- MODULO 278
- NBIN_VAL 279
- NEXT 265
- NO_FROM_LIST 279
- NOT_DEBUG 284
- NOT_INITIAL_LOAD 269
- NUMBER() 279
- NVL 279
- ORACLE 279
- Oracle-specific 262
- RAISE_EXCEPTION 280
- RENAME_OBJECT 280
- SELECT INTO_BEGIN 280
- SELECT INTO_BODY 280
- SMALLDATE 280
- SMALLINT 280
- SQLSERVER 280
- SSKEY 281
- SUBSTRING 281
- SUPERNVL 281
- TABLE_EXISTS_CONDITION 281
- TABLE_WITH_PREFIX 281
- TIMESTAMP_FILTER 268
- TIMESTAMP_FILTER_RANGE 268
- TINYINT 281
- TO_CHAR 281
- TO_DATE 282
- TO_DATEFMT 282
- TO_EPIDATE 282
- TO_NUMBER 282
- TO_TIME 282
- TO_YYYYMMDD 282
- TRANSLATE 283
- usage 264
- VAR 283
- VAR_ASSIGN_BEGIN 283
- VAR_ASSIGN_END 283
- VAR_ASSIGN INTO 283
- VARCHAR 283
- YYYYMMDD_FILTER 269
- YYYYMMDD_FILTER_RANGE 269
- YYYYMMDD_LAST_VALUE 269
- SQL operations 46
- SQL operators 120
 - COUNT 120
 - COUNT DISTINCT 120
 - SUM 120
- SQL Query dialog box 139
- SQL SELECT statements
 - fill from Query 139
- SQL Server. *See* Microsoft SQL Server
- SQL Statement dialog box 185, 186, 187
 - re-sizing 187
 - sample SQL 187
 - Tables References option 186, 304
 - Template feature 187, 304
 - zooming 187
- SQL statement extraction step 48
- SQL statements 46
 - error in 49
 - fact staging 307
 - populating staging tables 186, 303
 - pull/push 356
 - push/pull 48
 - setting breakpoints 79
 - stand-alone 48, 359
- SQLNET system call 261
- SQLSERVER SQL macro 280
- SRV log 343
- ss_keys 309
- SSKEY SQL macro 281
- sskeys 64, 65, 178, 305, 309, 312, 314, 315, 317, 318, 319, 322
 - duplicate 306
 - entering a pipeline stage 320
 - ikey 312, 318
 - new rows 312
 - UNKNOWN row 65
- Staging tables 39, 41, 49
 - truncating 100, 191, 198
- Star schemas 20, 39, 99, 306
- start_day_445 287
- start_year 287
- StorageManager log 246
- SUBSTRING SQL macro 281
- SUM. SQL operator 120
- SUPERNVL SQL macro 281
- SVGA monitors 190
- Symbol character, measure unit 289
- Synchronized groups 255
- System calls 53, 359
 - adding as job step 193
 - AGG 258
 - AGGVERIFY 259

INDEX

- APPSERVERHOST 259
 - APPSERVERPORT 259
 - CHARTSLOGFILE 259
 - CHARTSOUTPUTDIR 259
 - DATABASE 259
 - DBVENDOR 259
 - DEBUG_LEVEL 259
 - DIRNAME 259
 - DSN 260
 - EPIBIN 260
 - EXC 260
 - EXC_ARGS 260
 - EXC_CMD 260
 - EXCVERIFY 260
 - exit code 54
 - FILENAME 260
 - INSTANCE_NAME 260
 - INSTANCEROOTDIR 260
 - ISS 260
 - JOB_NAME 261
 - macro syntax 258
 - Momentum 189
 - PASSWORD 261
 - PATH 261
 - PROGRAM_NAME 261
 - Refresh 189
 - REGISTRY_EPIPATH 261
 - REGISTRY_ROOT 261
 - SERVER 261
 - SQLNET 261
 - USER 261
 - VERSION 261
 - SystemLogDir Registry key 242, 247
 - SystemLogDirWebpath Registry key 242
 - SystemLogWebDir Registry key 249
- T**
- TABLE_EXISTS_CONDITION SQL macro 281
 - TABLE_WITH_PREFIX SQL macro 281
 - Tables
 - base 347
 - metadata 323
 - rebuilding all 215
 - viewing updated 215
 - Tables, A and B. *See* A and B tables
 - Tablespaces, Oracle 262
 - alternate names for 263
 - Target
 - determining for Influence ticksheet 154
 - Influence ticksheet 153
 - TCP/IP sockets, SQL Server 336
 - TempDirGarbageLifetime Registry key 242
 - Template feature, SQL Statement dialog box 187
 - TemplateDir Registry key 242
 - TemplateServer 248
 - Ticksheet access
 - assigning 129
 - assigning to groups 203
 - assigning to users 207
 - Ticksheet dialog box 126, 129, 134, 141, 144, 145, 146
 - Cancel button 130
 - Ticksheet type label 293
 - Ticksheet types 293
 - Momentum Filter Households 171
 - Momentum Filter Individual 171
 - Ticksheets 14, 27
 - copying items 145
 - customizing labels 290
 - exporting 324
 - filters 134
 - glossary entries 134
 - grouped by dataset 213
 - metadata 324
 - modifying 176
 - option labels 290
 - saved reports 144
 - setting up attributes for 130
 - types 124
 - Time dimension (Date_0) 31
 - Time navigation 233
 - Time zone, displayed on reports 339
 - Time, uniform treatment of 31
 - TIMESTAMP_FILTER SQL macro 268
 - TIMESTAMP_FILTER_RANGE SQL macro 268
 - TIMESTAMP_LAST_VALUE 268
 - TIMESTAMP_LAST_VALUE SQL macro 268
 - Timestamps 71, 72, 74, 80
 - ignore option 72, 191
 - TINYINT SQL macro 281
 - TO_CHAR SQL macro 281

INDEX

TO_DATE SQL macro 282
 TO_DATEFMT SQL macro 282
 TO_EPIDATE SQL macro 282
 TO_NUMBER SQL macro 282
 TO_TIME SQL macro 282
 TO_YYYYMMDD SQL macro 282
 Toolbar icons, EpiCenter Manager 93
 Toplevel template 253
 Transaction Filter dialog box 174, 175
 Transaction filters 359
 defining Momentum 174–176
 Momentum 169
 Transaction type dimensions 98
 Transaction types 309, 359
 EpiCenter 288
 multiple 108
 Transactional data, uniform treatment of 32
 Transactional semantic types 317
 Transactional/State-like semantic type 318, 322
 Transactional/State-like/Force Close semantic type 319
 Transactions, tracking changes in inventory 33
 TRANSLATE SQL macros 283
 Transtype base dimensions 98
 Transtype dimensions. *See* Transaction type dimensions
 transtype_0 table 288
 transtype_key 309, 320
 Trends 125
 aggregates 151
 default ticksheet 148
 Truncate staging tables 100, 191
 Truncate Tables before Extraction dialog box 199
 Truncation
 external tables 198
 staging tables 198

U

Units measurement 120
 UNKNOWN
 dimension column 102
 rows 65, 223
 sskey 313
 string 64, 65
 Unregistering servers/EpiCenters 94
 Update button 288
 Update Unjoined. *See* Count Unjoined semantic type 316
 User dialog box 205
 USER system call 261
 Users
 assigning ticksheet access to 129
 setting up 205

V

VAR SQL macro 283
 VAR_ASSIGN_BEGIN SQL macro 283
 VAR_ASSIGN_END SQL macro 283
 VAR_ASSIGN_INT0 SQL macro 283
 VARCHAR SQL macros 283
 Verbose option, EpiChannel 78
 Version number, EpiCenter Manager 287
 VERSION system call 261
 VGA monitors 190

W

Web 249
 attributes as link to 132
 browser security considerations 199
 Web browser 13, 27, 210
 login dialog box 254
 Web Builder 87, 221, 233, 353
 measure terms 359

Web server 27, 225, 243, 245, 338, 340, 346
 alias 249
 authentication 250
 authentication method 250
 finding aliases 338
 instance_name alias 337
 log file location 242
 Object Not Found 340
 starting 239
 stopping 239
 URL accessing 340
 URL address 225
 Web server error message, Object Not Found 340
 Web URL address 225
 WebBuilder.exe command 360
 week_start_day 287
 Windows NT
 authentication 206, 231
 authentication module 250
 Authentication modules tips 253
 domain authentication 252
 group access rights 200
 group member synchronization 251
 importing users into groups 202
 Performance Monitor 81–82
 Working directory 58, 80
 location of 80
 World Wide Web (WWW). *See* Web.

Y

Years, beginning/end of EpiMart 216
 Yen sign 289
 YYYYMMDD_FILTER SQL macro 269
 YYYYMMDD_FILTER_RANGE SQL macro 269
 YYYYMMDD_LAST_VALUE SQL macro 269

INDEX

09625518 072500



E.PIPHANY E.4 INSTALLATION GUIDE

RELEASE 4.0

JUNE 1999

E.piphany Confidential

005270 07552900

E.PIPHANY E.4 INSTALLATION GUIDE

RELEASE 4.0

JUNE 1999

Epiphany Confidential

005270 01552960
00625510 072500

COPYRIGHT AND TRADEMARKS

Copyright © 1997-1999 by E.piphany, Inc. All rights reserved.

This document and the software it describes are furnished under license and may be used or copied only in accordance with such license. Except as permitted by such license, the contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, without the prior written permission of E.piphany, Inc.

This document contains propriety and confidential information of E.piphany, Inc. The contents of this document are for informational use only, and the contents are subject to change without notice. E.piphany, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

Unpublished rights reserved under the copyright Laws of the United States.

E.piphany™ and E.piphany e.4™ are trademarks of E.piphany, Inc. All other products or name brands are trademarks of their respective holders.

Printed in the USA

RESTRICTED RIGHTS LEGEND

Software and accompanying materials acquired with United States Federal Government funds or intended for use within or for any United States federal agency are provided with "Restricted Rights" as defined in DFARS 252.227-7013(c)(1(ii)) or FAR 52.227-19.

E.piphany Confidential

09625518 072500

TABLE OF CONTENTS

INTRODUCTION	vii
About E.piphany e.4	vii
New Features	viii
New Features for End Users	viii
Enhancements for Administrators	ix
Supported Platforms	x
About This Guide	x
About e.4 Documentation	xi
CHAPTER 1: PREPARING TO INSTALL E.PIPHANY SOFTWARE	13
Order of Installation	15
Installing and Configuring Your EpiCenter Host Computer	15
Installing and Configuring a Windows NT Server Host	16
Hardware Requirements for Windows NT Server	16
Windows NT Server 4.0 Installation	18
Disk-Volume Configuration	19
Network Connectivity	20
Installing and Configuring a Solaris Host	20
Hardware Requirements for Solaris	21
Solaris Installation	23
Configuration of Solaris Resources	23
Disk-Volume Configuration	25
Creating User and Group IDs	25

Creating Mount Points for Oracle Tablespaces	25
Network Connectivity	29
Installing and Configuring Your Database Server	30
Installing and Configuring SQL Server	30
Installing SQL Server	30
Configuring SQL Server	31
Assigning Data Stores in SQL Server	32
Installing and Configuring Oracle	34
Installing or Upgrading Oracle	34
Configuring Oracle	38
Creating Control Files	41
Assigning Data Stores in Oracle	43
Creating EpiMart and EpiMeta Users	44
Preparing Your Application Host	45
Installing Windows NT Server 4.0 on Your Application Host ..	45
Installing Internet Information Server, Version 4.0	45
Installing Microsoft Office Components	46
Installing and Configuring Connectivity Packages for Your	
Database Client Software	46
SQL Utilities	47
Oracle8 Utilities	47
CHAPTER 2: INSTALLING E.PIPHANY SOFTWARE	49
Performing the Installation	51
Application Server	52
Remote Administration	55
Configuring E.piphany Software	56
Configuring the E.piphany Web-server Proxy	56
Setting Up NT Performance Monitoring for Extraction	57
Verifying Your Installation	58

CHAPTER 3: PREPARING TO MIGRATE FROM	
PREVIOUS RELEASES	59
Migrating an EpiMart Database	59
SQL Server	60
Oracle	61
Exporting Metadata	61
Modifying Extraction Jobs	62

09625518.072500

09625518.072500

ABOUT E.PIPHANY

E.piphany is the industry leader in providing analytic applications that operate with Web-based efficiency and ease of use. Using the E.piphany e.4 System, you can integrate back-office and front-office solutions with a diverse range of third-party data sources. These sources include Enterprise Resource Planning (ERP), Customer Relationship Management (CRM), legacy, e-commerce, and front-office applications, along with a variety of external data providers. Because it has been built using open industry standards, the E.piphany e.4 System not only combines all relevant information, it also can interface with customer touchpoint systems such as Web, e-mail, sales-force automation, and call centers.

The E.piphany e.4 System provides quick access to your company's hard-earned store of information, which promotes stronger relationships with your most valuable customers. No matter which applications you have selected, E.piphany can help you build a 1-to-1 enterprise that serves the individual needs of your customers and delivers speed, flexibility and convenience for your day-to-day operations.

ABOUT E.PIPHANY e.4

The E.piphany e.4 System includes a sophisticated datamart, a set of advanced Web-based applications, and a highly configurable user interface. Using this integrated system, technical professionals can deploy customized and polished solutions for a variety of business problems.

The E.piphany datamart includes powerful and flexible data-management tools, including:

- Customizable extractors for a wide variety of data sources
- Built-in data-transformation semantics
- Advanced schema and metadata management
- Powerful performance-acceleration technologies

In addition, E.piphany provides a growing number of packaged solutions. With a minimum of customization, these solutions can help users solve a variety of business challenges using campaign management, on-line analytical processing (OLAP), data mining, and other activities that require integrated data from throughout your enterprise.

E.piphany solutions are easy to maintain and extend, allowing technical professionals greater scope and creativity in providing needed services.

NEW FEATURES

This release contains a number of new features that make E.piphany e.4 applications especially useful and easy to maintain.

NEW FEATURES FOR END USERS

- The Home page is now organized by topics, with links to your favorite reports. You can also display your favorite charts on the Home page.
- New navigation features include a global navigation bar and context-sensitive menus. The navigation bar allows you to find the topic you need. The context-sensitive links and numbered guidance icons that appear in each Web page, allow you to find the answers to your specific questions. Results automatically appear in the same window from which you make a request.

- New clustering, modeling and scoring applications allow you to create predictive models and use them to score lists and identify segments within those lists. For example, you can rank campaign prospects based on their likelihood to purchase. You can score lists by model or by measure, and create lift charts and profitability charts that help you project the potential returns from a campaign.
- A full-featured Campaign Manager is built into E.piphany e.4. You can define treatments, divide your list of potential recipients into segments, create control groups, and assign treatments to cells within each segment. You can download your list of recipients into output files by according to treatment. You can schedule a campaign job to run as soon as possible, on a later date, or on a repeating basis. To close the loop, you can feed the execution results back into the datamart for post-campaign analysis.
- There are no more “Cut and Paste” buttons. Filter settings and other parameters are carried forward implicitly and seamlessly from one Web page to another. For example, when you carry your list definition into an analysis report, the list is automatically converted into a filter.

ENHANCEMENTS FOR ADMINISTRATORS

- Incremental Nightly Processing delivers valuable efficiency and scalability. Rather than processing semantics and aggregates on all the data, you can limit operations to the increments that have changed. The time savings can be substantial.
- Sharable Web-page contents simplify much of your work. You can set up the filters, options, attributes, and measures (FOAM) just the way you want them, and then reuse those elements in a variety of Web pages.
- The Aggregate Optimizer helps you to identify the aggregates that will give your users the best response-times. You can find out which aggregates are being used, and which ones you can drop. You can also identify the aggregate that best covers a frequent user query and check its size before you add it.

- A growing list of packaged applications allow you to deploy new solutions quickly.

SUPPORTED PLATFORMS

This release of E.piphany e.4 supports the implementation of an EpiCenter datamart on the following database-server platforms:

- Oracle8 on Solaris 2.6 (Version 8.0.5.1)
- SQL Server 7.0, Enterprise Edition, on Windows NT Server 4.0 Enterprise Edition (with Option Pack 4 and the latest service pack)

For further information about this release of E.piphany e.4 software, refer to the Release Notes. To review the Release Notes after you have installed your E.piphany e.4 system, click the Microsoft Windows **Start** button, then choose **Programs**, then **E.piphany**, then the name of your Release 4.0 instance, then **Documentation**, and then **Release Notes**.

ABOUT THIS GUIDE

This guide is intended for database administrators and consultants who install, configure, and maintain E.piphany datamarts and solutions. Please follow these guidelines when you install E.piphany software.

1. Read Chapter 1 of this guide to ensure that you understand the following topics:
 - The considerations involved in preparing the host computer, the operating system (OS), the database server, and networking connectivity for your EpiCenter datamart
 - The steps that are required to prepare the host computer for your E.piphany application software

2. Make sure that you have the appropriate installation and configuration manuals on hand for the hardware and software products on which your Epiphany software depends. Epiphany suggests that you obtain the following manuals, depending upon the database-server option you select.
 - *Net8 Getting Started* (for use with Windows NT Server clients connected to Oracle/Solaris datamarts)
 - *Oracle8 Installation Guide for SunSPARC Solaris*
 - *Oracle8 Reference*
 - *Solaris 2.6 Hardware, SMCC Hardware Platform Guide*
 - *Solaris 2.6 SPARC Installation Instructions*
 - *SQL Server Administration Guide* (for Windows NT Server)
 - *Start Here: Basics and Installation, Microsoft Windows NT Server*
3. Please install the hardware and software products that are mentioned in this guide according to manufacturer instructions.
4. Please follow the configuration recommendations that are suggested in this guide.

If specific instructions do not appear for a particular configuration step or option, default values are acceptable.

ABOUT e.4 DOCUMENTATION

All of the manuals in the Epiphany e.4 documentation library are available for viewing on-line with Adobe Acrobat Reader. To view a manual in the documentation set, choose **Programs** from the Microsoft Windows **Start** menu, then **Epiphany**, then the name of your Release 4.0 instance, then **Documentation**, and then the title of your choice. To review additions and corrections that were inadvertently omitted from a manual, choose **Updates**.

E.piphany provides a full-text search index that allows you to perform keyword searches across all of the manuals in the documentation set. You must have Acrobat Reader with Search[®] or Acrobat Exchange[®] installed on your computer to use the index. To obtain a free copy of Acrobat Reader with Search, visit the following Web site:

<http://www.adobe.com/prodindex/acrobat/readstep.html>

To perform a keyword search in Acrobat Reader with Search or Acrobat Exchange:

1. Choose **Search** and then **Query** from the **Tools** menu
2. Enter a keyword or phrase to search for in the Keyword Search dialog box
3. Select the appropriate search options.
4. Click **Search**.
5. Choose a document in the Search Results dialog box.

Acrobat Exchange and Acrobat Reader with Search include tools that enable you to expand and limit your search criteria. For complete instructions on using the search capabilities of Adobe Acrobat, choose **Search Online Guide** from the **Help** menu.

PREPARING TO INSTALL E.PIPHANY SOFTWARE

The process of installing E.piphany software requires that you first install and configure hardware and software components that are supplied by other manufacturers. The success of your E.piphany application depends on the correct installation and proper configuration of these components.

This manual provides general installation guidelines and configuration recommendations for the products on which E.piphany software depends. For specific instructions about a particular product, please refer to the manufacturer's documentation. The configuration recommendations discussed in this manual apply to E.piphany enterprise-relationship-management (ERM) applications only.

The components that you must install and configure before installing E.piphany software include:

- A dedicated host computer for your EpiCenter datamart (strongly recommended) running one of the following operating systems:
 - Windows NT Server 4.0, Enterprise Edition with Service Pack 4 and the Microsoft Exchange mail-client utility from Options Pack 4
 - Sparc Solaris 2.6
- RAID disk-volume support (optional but recommended)
- A relational database server from the following list:
 - SQL Server 7.0 (E.piphany recommends the Enterprise Edition)
 - Oracle8, Release 8.0.5.1.0 for Solaris, with the Oracle Partitioning Option

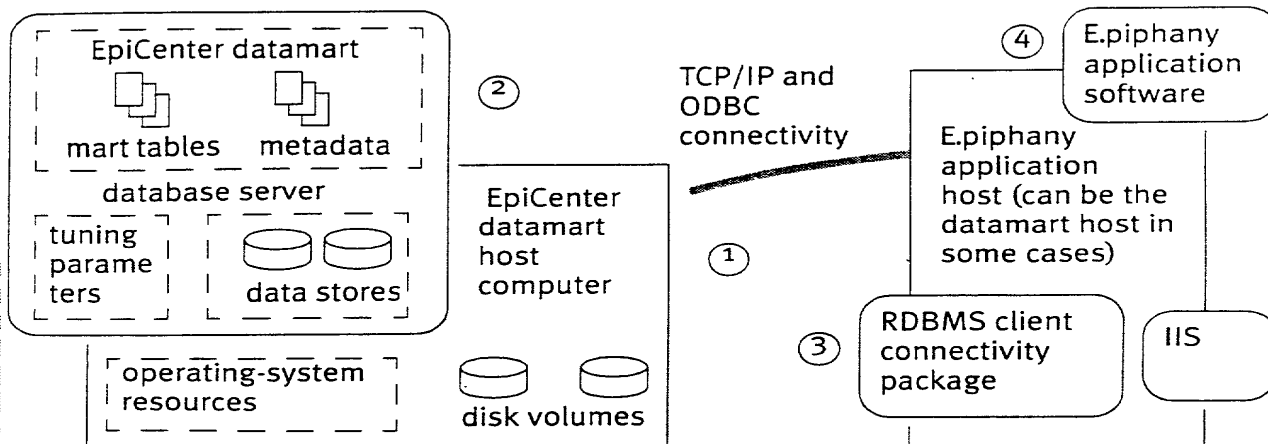
- A host computer for your E.piphany application running Windows NT Server
- Client utilities for your application host, including:
 - Microsoft Internet Information Server (IIS), Version 4.0, and the Microsoft Exchange Mail utility (available with Option Pack 4 for Windows NT Server)
 - The appropriate connectivity and management package:
 - SQL Server Utilities with a datamart that resides on SQL Server
 - Oracle8 Client for Windows with a datamart that resides on Oracle

Note: A separate application host is required only when you install your EpiCenter datamart on a Solaris host. If you install your datamart and E.piphany application software on the same computer, you must install the appropriate client utilities listed above on your datamart host.

ORDER OF INSTALLATION

Figure 1 illustrates the order in which you install and configure the components of your EpiPhany application.

FIGURE 1: EPIPHANY INSTALLATION COMPONENTS



INSTALLING AND CONFIGURING YOUR EPICENTER HOST COMPUTER

The specific installation and configuration tasks you must perform depend upon the host computer on which you intend to install your EpiCenter datamart. The sections that follow discuss the installation and configuration process for the following hosts:

- Pentium-based computers running Windows NT Server 4.0
- Sparc-based computers running Solaris 2.6

INSTALLING AND CONFIGURING A WINDOWS NT SERVER HOST

This section provides guidelines for installing and configuring a computer running Windows NT Server 4.0 as an EpiCenter host. See “Installing and Configuring a Solaris Host,” on page 20 for information about Solaris hosts.

HARDWARE REQUIREMENTS FOR WINDOWS NT SERVER

Please ensure that your computer meets the following minimum requirements:

- At least one CPU with a speed of 200 megahertz or higher (two or more CPUs are preferred), capable of supporting Windows NT Server 4.0, Enterprise Edition
Refer to the Microsoft manual entitled *Hardware Compatibility List; Microsoft Windows NT*, Version 4.0, for listings of computers that support Windows NT Server.
- The full complement of random-access memory (RAM) that your computer supports (no less than 128 megabytes)
- Depending on the amount of physical memory that you have installed, space of not less than two times the size of RAM
- A TCP/IP network controller installed and connected to a local area network (LAN)

Multiple network-card configurations are not currently supported.

- An adequate amount of disk space:
Epiphany software requires about 30 megabytes of disk space (including supporting software such as ODBC and JDBC drivers). This amount is in addition to the size of your datamart, which depends on the amount of data that you extract from your source systems. If you already have specifications for your datamart, you can use that information to estimate the size of your datamart data, taking into account the following considerations:

- The datamart includes two copies of the database and aggregates that are between three and ten times the base-table size, with additional space needed for staging tables, indexes, keys, and 100 megabytes of E.piphany metadata.
- E.piphany recommends that you reserve a separate log volume that is about 10 percent of the size of your datamart.
- E.piphany recommends that you reserve a volume that is at least twice the size of your largest fact table for temporary tables.

Based on these considerations and depending on the data storage hardware that your computer is equipped with, E.piphany suggests that you reserve the following disk volumes for your EpiCenter datamart:

- a RAID-1+0 volume for your EpiMart data and EpiMeta metadata

Use the following formula to determine the size of this volume:

$$\text{mart_vol_size} = 2 * (\text{aggs} * \text{fact_rows} * (100 + \text{fact_row_width})) + 2 * (\text{dimension_rows} * (20 + \text{dimension_row_width})) + 400 \text{ Mb}$$

Replace *aggs* with a value between 3 and 10 depending on the degree to which your application requires aggregate data. Use a higher value for applications that require extensive use of aggregate data.

- a RAID-1 volume for the log device associated with your datamart

$$\text{log_vol_size} = (0.1 * \text{mart_vol_size}) + 10\text{MB}$$

- a RAID-5 volume for the temporary database

Use the following formula to calculate the size of this volume:

$$\text{temp_db_size} = \text{tmp_factor} * \text{fact_rows} * (100 + \text{fact_row_width})$$

Replace *tmp_factor* with a value between 2 and 10, depending on your application.

E.piphany recommends that you format all disk volumes as NTFS drives, each with a block size of 64 kilobytes.

Note: The performance and costs of RAID volumes are heavily dependent on the specific hardware that you use. The documentation for your RAID equipment might include recommendations that differ from the suggestions made above. If so, E.piphany recommends that you follow the manufacturer's recommendations.

WINDOWS NT SERVER 4.0 INSTALLATION

You must install Microsoft Windows NT Server 4.0 Enterprise Edition, Service Pack 4 (or higher), and selected Microsoft Office components.

INSTALLING WINDOWS NT SERVER

To install Windows NT Server 4.0, follow the directions provided in Part 2 of the Microsoft manual entitled *Start Here: Basics and Installation, Microsoft Windows NT Server, Version 4.0*, and follow these recommendations:

1. Select the default directory location in which to install Windows NT Server.
2. Assign a computer name of 15 or fewer characters and write this name down for future use.
3. Indicate the server type as a stand-alone server.
4. Enter your administrator account name and password.
5. Create a repair disk in case of an emergency.
6. Select the default list of components.

As part of the Windows NT Setup, after you finish setting up the network parameters, the Finishing Setup dialog box is displayed, which informs you that are about to start setting up Microsoft Internet Information Server, Version 2.0. Exit the installer at this time. You do not need to install IIS, Version 2.

Note: When installing the operating system, you might have to install additional drivers for 3rd-party disk drives, graphics cards, or other hardware.

INSTALLING OPTION PACK 4

Detailed instructions for installing Option Pack 4 are provided on the installation CD-ROM. Please follow those instructions to install Microsoft Internet Information Server, Version 4.x.

INSTALLING SERVICE PACK 4 (OR HIGHER)

Detailed instructions for installing your service pack are provided on the installation CD-ROM. Please follow those instructions to install the service pack.

Note: You must install Option Pack 4 before you install the latest Service Pack.

INSTALLING MICROSOFT OFFICE COMPONENTS

You can install Microsoft Office from the Microsoft Office CD. The installation program provides detailed instructions. Please follow those instructions to install the Microsoft Exchange messaging client, ODBC connection client, and any other components of Microsoft Office that you choose to include.

DISK-VOLUME CONFIGURATION

Take the following steps to configure disk volumes for your datamart:

1. The disk volumes for your database server use NTFS format. Some computer models running Windows NT Server configure disks with FAT format by default. To convert disks from FAT to NTFS, use the Disk Administrator. From the Start Menu, choose Programs, then Administrative Tools, and then Disk Administrator.
2. Refer to the disk-volume size figures that you calculated using the formulas in "Hardware Requirements for Windows NT Server," on page 16 for size and RAID-level information.

3. If you are using a hardware RAID controller or RAID software, follow the manufacturer's instructions for configuring your disk volumes. Otherwise, use the Disk Administrator to configure your disk volumes.
4. In a command-window prompt, enter the following command to enable monitoring of disk I/O performance with the PerfMonitor utility:

```
diskperf -y
```

NETWORK CONNECTIVITY

Epiphany software requires the TCP/IP network protocol. Please contact your network administrator for specific instructions regarding the installation and configuration of devices, drivers, and domains for this protocol.

When you have verified that you have network connectivity, you can proceed to the next stage of preparations. Refer to "Installing and Configuring SQL Server," on page 30 for further instructions.

INSTALLING AND CONFIGURING A SOLARIS HOST

This section provides guidelines for installing and configuring a computer running Solaris 2.6 as an EpiCenter host. Please review this section even if you plan to install your E.piphany datamart on an existing Oracle host.

This section discusses a number of topics involved in installing and configuring Solaris for use with Oracle8. For additional information, refer to the *Oracle8 Installation Guide for SunSPARC Solaris* and *Oracle Support Bulletin 104511.69*.

For information about a Windows NT Server host, see "Installing and Configuring a Windows NT Server Host," on page 16.

HARDWARE REQUIREMENTS FOR SOLARIS

Please ensure that your computer meets the following minimum requirements:

- At least one CPU with a speed of 200 megahertz or higher (two or more CPUs are preferred), capable of supporting Solaris 2.6
- The full complement of random-access memory (RAM) that your computer supports (no less than 128 megabytes)
- Depending on the amount of physical memory that you have installed, space of not less than two times the size of RAM
- A Solaris-supported CD-ROM drive that uses High Sierra or ISO 9660 format with the Rockridge extension
- A TCP/IP interface installed and connected to a local area network (LAN)
- An adequate amount of disk space:

Epiphany software requires about 30 megabytes of disk space (including supporting software such as ODBC and JDBC drivers). This amount is in addition to the size of your datamart, which depends on the amount of data that you extract from your source systems. If you already have specifications for your datamart, you can use that information to estimate the size of your datamart data, taking into account the following considerations:

- The datamart includes two copies of the database and aggregates that are between three and ten times the base-table size, with additional space included for staging tables, indexes, keys, and 100 megabytes of E.piphany metadata.
- E.piphany recommends that you provide space for temporary tables that is at least twice the size of your largest fact table, and space for rollback segments that 10 percent of the size of your datamart.
- E.piphany recommends that you use separate devices for redo logs and archived log files.

Based on these considerations, E.piphany suggests that you use the following formula to calculate the size of your datamart:

```
data_size = 2 * (aggs * fact_rows * (100 + fact_row_width))
            + 2 * (dimension_rows * (20 + dimension_row_width)) + 400MB
            + (0.1 * mart_vol_size) + 10MB
            + tmp * fact_rows * (100 + fact_row_width)

rollback_seg_size =      mart_size / 5

mart_size =      data_size + rollback_seg_size
```

Replace *aggs* with a value between 3 and 10, depending on the degree to which you expect your application to require aggregate data. Use a higher value for applications that require extensive use of aggregate data. Replace *tmp* with a value between 3 and 10, depending on the degree to which you expect your application to require temporary tables.

E.piphany recommends that you configure a RAID 1+0 array as the disk volume for your datamart data. For enhanced performance, you might choose to spread datamart data across two or more such disk volumes, depending on the storage devices and controllers that are available to you.

E.piphany also recommends that you configure the following additional disk volumes:

- A volume for the Oracle installation files, initialization files, and redo logs
- A volume for Oracle system tables and log archives

In keeping with the Optimal Flexible Architecture guidelines for Oracle, E.piphany recommends choosing the following volume names:

- **/u01** for system rollback segments
- **/u02** for system tables
- **/u03** for datamart data

If you configure additional disk volumes for datamart data, E.piphany suggests that you begin the numbering scheme for those volume-name suffixes with 4 (**/u04**).

Note: The performance and costs of RAID volumes are heavily dependent on the specific hardware that you use. The documentation for your RAID equipment might include recommendations that differ from the suggestions made above. If so, E.piphany recommends that you follow the manufacturer's recommendations.

SOLARIS INSTALLATION

Follow the directions provided with your Solaris software to install the latest Solaris operating system and current patch level from Sun Microsystems.

CONFIGURATION OF SOLARIS RESOURCES

You must configure the following Solaris resources to support an Oracle database server for your EpiCenter datamart:

- Semaphores, shared-memory segments, and the paging threshold in the **/etc/system** file
- Disk volumes for Oracle data
- User **oracle**, group **dba**, and user **epichnl**
- Mount points for Oracle tablespaces, redo logs, and log archives
- Automatic start-up and shut-down for Oracle
- Network Connectivity

The sections that follow describe these activities.

UPDATING THE /ETC/SYSTEM FILE

As **root**, you must edit the **/etc/system** file to configure the following system resources for Oracle:

- Semaphores
- Shared memory
- Swapping threshold

Increasing values for these parameters allocates more resources, which reduces the amount of physical memory that is available to processes. Add or modify the following lines in `/etc/system` to configure these resources:

```
set semsys:seminfo_semmni=70
set semsys:seminfo_semmsl=100
set semsys:seminfo_semmns=200

set shmsys:shminfo_shmseg=10
set shmsys:shminfo_shmmni=100
set shmsys:shminfo_shmmin=1
set shmsys:shminfo_shmmax=max_shared_size

set lotsfree=pages
```

1. Update the values for the `semmni`, `semmsl`, `semmns`, `shmseg`, `shmmni` and `shmmin` parameters as indicated above. If you are adding an Oracle instance to a host on which another instance already resides, multiply the values indicated above for each parameter by the number of instances that are to be supported on this host.
2. Add a new line to set the `shmax` resource. Replace `max_shared_size` with a value that is between 80 percent and 90 percent of the size of physical memory in bytes. This parameter indicates a total allowable size for shared memory; it does not allocate any memory resources.
3. Update the value for the `lotsfree` parameter. Replace `pages` with the minimum number of pages that your computer requires in order to prevent unnecessary swapping. E.piphany recommends a value between 2 percent and 6 percent of the total number of pages in RAM. Use the following formula calculate the number of 8-kilobyte pages in RAM:

$$\text{total_pages} = \text{megs} * 128$$

Replace *megs* with the number of megabytes of RAM that are installed on your computer.

4. Use the **reboot** command to reboot your Solaris host so that these semaphore and shared-memory settings can take effect.

DISK-VOLUME CONFIGURATION

As **root**, take the following steps to configure disk volumes for your datamart:

1. Refer to the disk-volume size figures that you calculated using the formulas in “Hardware Requirements for Solaris,” on page 21 for size and RAID-level information.
2. If you are using a hardware RAID controller or RAID software, follow the manufacturer’s instructions for configuring your disk volumes. Otherwise, use the standard Solaris utilities for configuring disk partitions.

CREATING USER AND GROUP IDS

As **root**, take the following steps to create the **oracle** and **epichnl** user IDs and the **dba** group, if those IDs are not already present.

1. Use **admintool** or the **useradd** utility to create the **oracle** and **epichnl** user IDs and the **dba** group.
2. Set the default group ID for both **oracle** and **epichnl** to **dba**.
3. Propagate this account and group information to the Network Information Service (NIS).

CREATING MOUNT POINTS FOR ORACLE TABLESPACES

As **root**, take the following steps to create mount points for Oracle:

1. Use the **mkdir** command to create mount points with names of the following form on the disk volumes that you have reserved for Oracle:

```
mkdir /u01/u02 /u03 /u04
```

These mount points correspond to the Optimal Flexible Architecture (OFA) specification for an Oracle database server. The mount points are used by Oracle as follows:

- **/u01** Oracle system tables
- **/u02** system rollback segments

- /u03 datamart tablespaces
 - /u04 additional datamart tablespaces (optional)
2. Use the following commands to assign user ownership, group ownership, and access permissions to each of these mount points.

```
chown oracle /u*
chgrp dba /u*
chmod 775 /u*
```

MODIFYING THE SHELL INITIALIZATION FILES

Take the following steps to modify the shell initialization files for the **oracle** and **epichnl** user IDs.

1. Log in as user **oracle**.
2. Edit the **.profile** file in the **oracle** home directory to add the following lines, which you can copy from the **/cdrom/resources/profile.sh** file on the E.piphany installation CD:

```
#DISPLAY=hostname:0.0# Uncomment this line and replace hostname
                        # with the name of the
                        # installation host.

ORACLE_BASE=/opt/oracle # Replace with actual pathname.
ORACLE_TERM=$TERM      # (Optional) Replace $TERM with
                        # alternate terminal type for Oracle.

MNT1=/u01              # Oracle system tables
MNT2=/u02              # Oracle system rollback segments
MNT3=/u03              # Datamart tables
MNT4=/u04              # Additional datamart tables (optional)

ORACLE_SID=ORCL        # Replace ORCL with actual SID.
ORACLE_HOME=/ $ORACLE_BASE/product/8.0.5
PATH=.: $ORACLE_HOME/bin: $ORACLE_BASE/local:/bin:/usr/bin:\
/usr/ccs/bin:/usr/openwin/bin:/usr/5bin:/usr/ucb
LD_LIBRARY_PATH=$LD_LIBRARY_PATH: $ORACLE_HOME/lib
TNS_ADMIN=$ORACLE_HOME/network/admin
ORA_NLS33=$ORACLE_HOME/ocommon/nls/admin/data
export ORACLE_BASE ORACLE_SID ORACLE_HOME ORACLE_TERM PATH \
DISPLAY LD_LIBRARY_PATH TNS_ADMIN ORA_NLS33 MNT1 MNT2 MNT3 MNT4
```



```
ORAENV_ASK=no
umask 022
```

3. Make the following edits to the **.profile** file:
 - a) If you are installing from a remote host, uncomment the line that sets the DISPLAY environment variable and replace *hostname* with the name of the remote host.
 - b) If the TERM variable is not set in your environment, or if you want to use a terminal type for Oracle utilities that differs from the default terminal type for your shell, replace \$TERM with the appropriate value for that terminal type.
 - c) Be sure that the pathname for the ORACLE_BASE environment variable is correct.
 - d) Set the ORACLE_SID environment variable to the correct instance name for your EpiCenter instance.
4. If you use the C shell, edit the **.cshrc** file to add the following lines, which you can copy from the **/cdrom/resources/cshrc.csh** file of the installation CD. Perform the same edits to this file that you did in the previous step. EpiPhany recommends that you update the **.cshrc** file in addition to, rather than instead of, the **.profile** file, so that users of any common shell can have environment variables set up automatically.

```
#setenv DISPLAY hostname:0.0# Uncomment this line and replace
# hostname with name of host for
# remote installation.

setenv ORACLE_BASE /opt/oracle # Match pathname in .profile.
ORACLE_TERM=$TERM # (Optional) Replace $TERM with
# alternate terminal type for Oracle.

setenv MNT1 /u01 # Oracle system tables
setenv MNT2 /u02 # Oracle system rollback segments
setenv MNT3 /u03 # Datamart tables
setenv MNT4 /u04 # Additional datamart tables (optional)

setenv ORACLE_SID ORCL# Replace ORCL with the actual SID.
setenv ORACLE_HOME $ORACLE_BASE/product/8.0.5
setenv PATH .:$ORACLE_HOME/bin:$ORACLE_BASE/local:/bin:\
/usr/bin:/usr/ccs/bin:/usr/openwin/bin:/usr/5bin:/usr/ucb
setenv LD_LIBRARY_PATH $LD_LIBRARY_PATH:$ORACLE_HOME/lib
```

```

setenv TNS_ADMIN $ORACLE_HOME/network/admin
setenv ORA_NLS33 $ORACLE_HOME/ocommon/nls/admin/data
set ORAENV_ASK=no
umask 022

```

5. Log in as user **epichnl** and copy the **.profile** and **.cshrc** files that you just edited from the **oracle** home directory to the home directory for **epichnl**.
6. Enable remote shell access for user **epichnl** by adding the following line to the **.rhosts** file in the **epichnl** home directory:

```

application_host      epichnl

```

Replace *application_host* with the hostname of the Windows NT Server host computer on which you intend to install your E.piphany application. Entries in the **.rhosts** file are case sensitive, so be sure that you enter the hostname exactly as it appears in the NT domain.

SETTING UP AUTOMATIC START-UP AND SHUT-DOWN FOR ORACLE

Take the following steps to set up automatic start-up and shut-down for Oracle:

1. Log in as **root**.
2. Enter the following command lines into the **/etc/init.d/dbora** file. You can copy this file from the **/cdrom/resources/dbora.sh** file of the installation CD. Replace *oracle_home_path* with the value of **ORACLE_HOME** as it appears in the **.profile** file that you created in the previous section.

```

ORACLE_HOME=oracle_home_path
ORACLE_OWNER=oracle
if [! -f $ORACLE_HOME/bin/dbstart -o ! -d $ORACLE_HOME]
then
    echo "oracle startup: cannot start"
    exit
fi
case "$1" in
'start')
    su - $ORACLE_OWNER -c $ORACLE_HOME/bin/dbstart &
;;

```

```
'stop')
    su - $ORACLE_OWNER -c $ORACLE_HOME/bin/dbshut &
;;
esac
su - $ORACLE_OWNER -c "lsnrctl start"
```

3. Create links to the **dbora** file in the system-initialization run-level directories:

```
ln -s /etc/init.d/dbora /etc/rc0.d/K10dbora
ln -s /etc/init.d/dbora /etc/rc2.d/S99dbora
```

4. Create a symbolic link for the **listener.ora** file:

```
ln -s /var/opt/oracle/listener.ora /etc/listener.ora
```

NETWORK CONNECTIVITY

The following steps enable network access for the Oracle8 database server:

1. Enable remote shell access from your E.piphany application host by adding an entry of the following form to the DNS hosts domain or the **/etc/hosts** file:

```
IP_address hostname
```

Replace *IP_address* with the IP address of the application host. Replace *hostname* with the hostname of the application host.

2. Add the following entry to the **/etc/services** file for your Oracle database server. If you are adding an Oracle instance to a host that already has one, you must add an additional **listener** entry for the new instance with a distinct port number.

```
listener      port/tcp      #TNS Listener
```

Replace *port* with the port number for your Oracle database server or Net8 service. E.piphany suggests port 1521 if you do not already have that port assigned.

3. Log in as a user other than **root**. Then use the **ftp** command to verify that your Solaris host is connected to the network and can transfer data.

When you have verified that you have network connectivity, you can proceed to the next stage of preparations. See “Installing and Configuring Oracle.” on page 34 for further instructions.

INSTALLING AND CONFIGURING YOUR DATABASE SERVER

This section provides guidelines and recommendations for:

- Installing and configuring the database server for your EpiCenter datamart
- Assigning data stores for your EpiCenter data
- Creating databases for your EpiCenter datamart and metadata

INSTALLING AND CONFIGURING SQL SERVER

This section provides guidelines and recommendations for installing and configuring SQL Server for use with an EpiCenter datamart. You must install the Enterprise Edition of SQL Server Version 7.0. This Edition can be installed only on hosts that are running the Enterprise Edition of Windows NT Server.

INSTALLING SQL SERVER

To install SQL Server, follow the directions provided on the installation CD, along with the following recommendations.

1. Choose Install Prerequisites from the initial installer dialog, then install Internet Explorer 4.01, Service Pack 1.
2. Choose Install SQL Server 7.0 Components from the initial dialog.
3. In the Install SQL Server 7.0 Components dialog, choose Enterprise Edition.
4. In the Select Install Method dialog, choose Local Install.

5. In the Convert Existing SQL Server Data dialog, check Yes, run the SQL Server Upgrade Wizard.
6. In the Setup Type dialog, choose the default values (Typical installation, default destination folders for program files and data files)
7. In the Service Accounts dialog, choose:
 - Use the same account for each service
 - Auto Start SQL Server Service
 - Use the Local System Account
8. If you are asked to choose a character set, choose 850 Multilingual.
9. If you are asked to choose a collating sequence, choose Unicode.
10. If you are asked to choose a sort order, choose Dictionary Order, Case Insensitive.
11. If you are asked to choose networking protocols, include TCP/IP sockets and the default port number.
12. If you are prompted to run the Upgrade Wizard, answer Yes, then follow the instructions in the Upgrade Wizard.

CONFIGURING SQL SERVER

Configuring SQL Server 7.0 involves the following actions:

- Registering and starting the server
- Configuring parallelism

REGISTERING AND STARTING SQL SERVER

To register you server and start SQL Server, take the following steps:

1. From the Start menu, choose Programs, then Microsoft SQL Server, and then Enterprise Manager.
2. Expand the SQL Server Group folder, then right-click the icon for your server.

3. Choose New Server Registration to bring up the Register SQL Server Wizard.
4. In the Select an SQL Server dialog, enter the name of your server in the text box above the list of available servers, then click Add.
5. In the Select Authentication Mode dialog, choose SQL Server authentication.
6. In the Select Server Group dialog, choose SQL Server Group.
7. Continue with this wizard to completion.
8. Expand the SQL Server Group folder to see your new server icon.
9. Right-click the icon for your server, then click Start.

CONFIGURING PARALLELISM

To configure parallelism, take the following steps:

1. Right-click the icon for your server, then choose Properties.
2. In the Server Properties dialog, choose the Processor tab.
3. Check the Boost SQL Server priority on Windows NT box.
4. Do not check the Use Windows NT Threads box.
5. If your host is dedicated exclusively to your datamart and you plan to run your E.piphany application on another computer, click the Use all available processors button. If you plan to run your E.piphany application on this same host, click the Use ... processor(s) button. Enter one less than the number of CPUs that reside on your host.

ASSIGNING DATA STORES IN SQL SERVER

Use the SQL Server Manager to set up new database devices and new databases in your EpiCenter datamart. You need to set up the following databases:

- A database for your datamart data

- A database for E.piphany metadata
- A temporary database

You must also create the transaction logs associated with each of these databases. E.piphany recommends that you place transaction logs on different disks than those on which your databases reside.

Take the following steps to set up each database:

1. Expand the folder for your database server in the Enterprise Manager.
2. Right-click on the Databases folder, then choose New Database.
3. Enter the following information in the General tab of the Database Properties dialog:
 - The name of the new database
 - The location
Click the button next to the entry for your new database in the Location column to browse for a location.
 - Automatic file growth in 10-megabyte increments
 - Unrestricted file growth
4. Enter the following information in the Transaction Log tab:
 - The location (click the button and browse)
 - Automatic file growth in 5-megabyte increments
 - Unrestricted file growth
5. Right-click the icon for you new database, then choose Properties.
6. In the Options tab of the Properties dialog, check only the following options. Uncheck any other options that might have been selected by default.
 - Select into / bulk copy
 - Truncate log on checkpoint
 - Auto create statistics

When you have completed these steps for each of the three databases, you can proceed to "Preparing Your Application Host," on page 45.

INSTALLING AND CONFIGURING ORACLE

This section provides guidelines and recommendations for installing and configuring Oracle8 for use with an EpiCenter datamart. Please review these sections even if you plan to use an existing Oracle instance to house your EpiCenter datamart.

INSTALLING OR UPGRADING ORACLE

The installation process for Oracle takes place in the following stages. Please complete each stage before proceeding to the next one:

- Installing or upgrading Oracle Version 8.0.5.0
- Installing the Oracle 8.0.5.1.0 (or higher) patch release
- Running the **root.sh** script (new installations only)

INSTALLING OR UPGRADING VERSION 8.0.5.0

If you have not already done so, please follow the directions provided in the *Oracle8 Installation Guide* to install Oracle or upgrade your Oracle instance to Version 8.0.5.0. The following steps summarize the installation process for this Oracle release.

1. Log in as user **oracle**.
2. Enter the following command to enable access to the window system on the local host. Replace *hostname* with the name of the local computer.

```
xhost hostname
```
3. Use one of the following commands to assign the local host as the display device for the Oracle installation GUI:

```
DISPLAY=hostname:0.0 ; export DISPLAY    # for sh or ksh
setenv DISPLAY hostname:0.0             # for csh
```


4. Change directory (**cd**) to **/cdrom/oracle805/orainst**, and enter one of the following commands to start the Oracle installer.


```

./orainst /m      # for the Motif-based graphical interface
./orainst /c      # for the character-terminal-based interface
      
```
5. If Oracle has not yet been installed, choose Default Install, then Install, Upgrade, or Deinstall Software, and then Install New Product with Database Objects. If Oracle has been installed, choose Add/Upgrade Software.
6. Confirm the pathnames of the ORACLE_BASE and ORACLE_HOME directories, and Oracle instance name, which is the value of the ORACLE_SID environment variable.
7. Confirm the location of the installation-log files.
8. Select the following products to install. Some options reside within categories in the Oracle Software Asset Manager dialog. The names of these categories, such as Oracle 8.0.5 Options, are prefixed with a plus sign (+). To expand a category and select from among the options it contains, double-click the name of that category. Then select the options of your choice.
 - Net8 Version 8.0.5.0.0
 - Net8 External Naming Adapters (entire category)
 - Net8 Protocol Adapters (entire category)
 - Oracle Advanced Networking Version 8.0.5.0.0 (entire category)
 - Oracle On Line Text Viewer
 - Oracle Partitioning Option (in the Oracle Options category)
 - Oracle UNIX Installer
 - Oracle8 Enterprise RDBMS 8.0.5.0.0
 - PL/SQL 8.0.5.0.0
 - Solaris Documentation
 - SQL*Plus

You can ignore any warning messages that appear regarding previously installed storage-manager products.

9. Choose **OK** when you are asked to confirm default database start-up.
10. When you are prompted to do so, enter the OFA-compliant mount points that you have defined for the datamart data, redo logs, and archive logs:

```

/u01          # Oracle system tables
/u02          # Oracle system rollback segments
/u03          # Datamart tables
/u04          # Additional datamart tables (optional)

```

11. Enter a suitable pathname for Oracle documentation when prompted. E.piphany suggest entering a pathname of the form:

```
/ORACLE_HOME/doc
```

Replace *ORACLE_HOME* with the pathname of the home directory for your Oracle instance.

Note: At this point, the Oracle installer begins to download files. This process typically takes less than an hour, and requires periodic checking for status or error messages. The progress bar that the installer displays does not extend fully across the box even though all files have been downloaded. This is not an error. You must take the steps that follow to complete the Oracle installation procedure.

12. When the installer displays the Information dialog box, choose **OK** in it and all subsequent dialog boxes, then exit the installer.
13. If you are adding a new Oracle instance to a host on which Oracle already resides, back up the **listener.ora** file and the **tsnames.ora** file, then edit these files to include entries for your new instance.

INSTALLING THE ORACLE 8.0.5.1.0 (OR HIGHER) PATCH

Take the following steps to install the Oracle 8.0.5.1.0 (or higher) patch:

1. Log into the Download section of the Oracle MetaLink Web site, then enter the following information. (This is a password-protected site. Contact your Oracle sales representative to obtain access to this site.)
 - Product: Oracle RDBMS
 - Platform: Sun SPARC Solaris
 - Sort by: Last update date
2. Choose patch set 80510 (or higher) from the list of available packages.
3. Follow the instructions that are provided in the accompanying README file to download and install the patch.

RUNNING THE ROOT.SH SCRIPT

If you are installing an Oracle8 database server for the first time, follow the steps below to run the **root.sh** script. Otherwise, you can proceed to the next section.

Warning: If you have upgraded from an earlier version of Oracle8, stop. Do not proceed with the following steps. Instead, continue with the next section, "Configuring Oracle," on page 38.

1. Log in as **root** and enter one of following shell commands to clear environment variables that might adversely affect the installation process:

```
unset SRCHOME TMPDIR TWOTASK      # Bourne or Korn shell

unsetenv SRCHOME TMPDIR TWOTASK   # C shell
```

2. Review the **root.sh** script, and if it is correct, run it. If not, you can update this script and then run it without having to rerun the installer:

```
cd $ORACLE_HOME/orainst
sh ./root.sh
```

3. Answer Y to the following prompt if it appears:

```
ORACLE_HOME does not match the home directory for Oracle.
OK to continue? [N]:
```

*Note: At this point, you might see a warning to the effect that the **ulimit** has been exceeded. You can ignore this warning.*

CONFIGURING ORACLE

Epiphany suggests that you take the following steps to configure Oracle8 to support an EpiCenter datamart. For detailed configuration instructions for Oracle, refer to the chapter entitled “Configuring the Oracle8 System” in the *Oracle8 Installation Guide*.

1. Log in as **oracle**.
2. Download the Epiphany configuration scripts for Oracle, as follows:
 - a) Insert the Epiphany installation disk in the CD-ROM drive on your Solaris host.
 - b) Enter the following commands:

```
cd $ORACLE_HOME/rdbms/admin
cp /cdrom/resources/* .
cd $ORACLE_BASE/admin/$ORACLE_SID/pfile
cp $ORACLE_HOME/rdbms/admin/init$ORACLE_SID.ora .
```

3. Ensure that the Oracle instance for your EpiCenter datamart is running:

```
/bin/ps -ef | grep pmon
```

If the instance is running, the **ps** command displays a process listing that includes the SID for your Oracle instance.

If your instance is *not* running, take the following steps to start it:

- a) Enter the **svrmgrl** command:

```
svrmgrl # starts the server manager
```

b) Enter the following commands in **svrmgrl**:

```
connect internal
startup
disconnect
exit
```

4. Check to see that the Oracle listener process is running by entering the following command:

```
lsnrctl status
```

If this process is not running, **lsnrctl** displays a number of “unable to connect” and “no listener” messages, among others. To start the listener process, enter the following command:

```
lsnrctl start
```

5. Verify that there is an entry for your EpiCenter database in the **/var/opt/oracle/oratab** file of the following form. If you are adding an instance to a host on which Oracle already resides, add an entry for the new instance in addition to the entries for any previous instances that might already be present.

```
ORACLE_SID:ORACLE_HOME: Y
```

Replace *ORACLE_SID* with the value of the *ORACLE_SID* environment variable. Replace *ORACLE_HOME* with the value of the *ORACLE_HOME* environment variable.

Note: If the letter N appears at the end of the entry, replace it with Y to enable automatic start-up whenever the operating system reboots.

6. Make sure that the Oracle command file has correct permissions, as follows:

```
cd $ORACLE_HOME/bin
chmod 4751 oracle
ls -lg oracle
```

The correct permissions are:

```
-rwsr-x--x  oracle  dba  oracle
```

7. Enter the following commands to create symbolic links to the **\$ORACLE_HOME/dbs/init\$ORACLE_SID.ora** file for your Oracle instance.

```
cd $ORACLE_HOME/dbs
mv init$ORACLE_SID.ora init$ORACLE_SID.orig
ln -s ../rdbms/admin/init$ORACLE_SID.ora init$ORACLE_SID.ora
ln -s ../rdbms/admin/init$ORACLE_SID.ora \
    $ORACLE_BASE/admin/$ORACLE_SID/pfile/init$ORACLE_SID.ora
```

8. Edit your **init\$ORACLE_SID.ora** file to provide values that are consistent with the size of your application by following the instructions that are embedded in that file. Please refer to the *Oracle8 Reference* for detailed information about mount points, initialization parameters, and values.

Note: E.piphany applications typically perform large decision-support queries, as opposed to the large numbers of concurrent-but-brief queries of a typical on-line-transaction-processing (OLTP) application. E.piphany recommends that you configure Oracle using initialization parameters and values that are geared toward high TPC-D (Transaction Processing Performance Council benchmark D) performance. The sample file provides suggested values only, which E.piphany recommends that you edit to suit your application. You can adjust those parameters over time as you learn more about performance in your specific circumstances.

9. Enter the following commands to create symbolic links to the **\$ORACLE_HOME/rdbms/admin/config\$ORACLE_SID.ora** file:

```
ln -s ../rdbms/admin/config$ORACLE_SID.ora \
    config$ORACLE_SID.ora
ln -s ../rdbms/admin/config$ORACLE_SID.ora \
    $ORACLE_BASE/admin/$ORACLE_SID/pfile/config$ORACLE_SID.ora
```

10. If you are configuring an Oracle instance on a host on which Oracle already resides, enter the following commands to create administrative directories for your new instance:

```
mkdir $ORACLE_BASE/admin/$ORACLE_SID
mkdir $ORACLE_BASE/admin/$ORACLE_SID/bdump
mkdir $ORACLE_BASE/admin/$ORACLE_SID/cdump
mkdir $ORACLE_BASE/admin/$ORACLE_SID/pfile
mkdir $ORACLE_BASE/admin/$ORACLE_SID/vdump
```

CREATING CONTROL FILES

Epiphany provides a sample SQL script, called **epi_idb.sql**, that you must edit and then run to create the control files and perform other initialization tasks. Step 2, on page 38 describes how to download this and other sample scripts.

```
-- epi_idb: create system tables, control files, and rollback
segments
-- Replace all occurrences of 'SID' with the SID of your Oracle
instance.
-- Be sure to edit pathnames. Names are case sensitive.

connect internal;
set echo on;
set termout on;
spool dbcre.log;
shutdown;

-- initialization file
startup pfile=/u01/app/oracle/admin/SID/pfile/initSID.ora nomount
select to_char(sysdate, 'MM-DD-YYYY HH24:MM:SS') now from dual;
create database
  controlfile reuse

  -- redo log files:
logfile '/u02/app/oradata/SID/redoSID01.log' size 100m reuse,
        '/u02/app/oradata/SID/redoSID02.log' size 100m reuse

  -- system tablespace file:
datafile '/u01/oradata/SID/system01.dbf' size 180m reuse
maxdatafiles 100;
create public rollback segment RS1
storage(initial 300K next 300k);
```

```

alter rollback segment RS1 online;
shutdown;

-- startup pfile:
startup pfile=/u01/app/oracle/admin/SID/pfile/initSID.ora
select to_char(sysdate, 'MM-DD-YYYY HH24:MM:SS') now from dual;
spool off;

@catalog.sql;
@catparr.sql;
@catproc.sql;
@utlxplan.sql;
@catldr.sql;

exit;
!EOF

spool off;

```

To edit this script, you must replace every occurrence of *SID* with the SID of your Oracle instance. Please be aware that quoted strings and pathnames are case sensitive in Oracle SQL. To avoid case-sensitivity problems, E.piphany tablespace names and filenames are all uppercase.

When you have edited the script, you can run it using **svrmgrl**, as follows:

1. Log in as **oracle**, then enter the following shell commands:

```

cd $ORACLE_HOME/rdbms/admin
svrmgrl

```

2. When you see the **svrmgrl** prompt, enter the following commands:

```

connect internal;
@epi_idb.sql
exit

```

Note: The **epi_idb.sql** script calls several system-catalog-creation scripts that produce copious status messages and “drop object” warnings that you can ignore.

ASSIGNING DATA STORES IN ORACLE

EpiPhany provides a UNIX (Bourne) shell script, called **epi_tsp.sh**, that you can use as an aid in configuring tablespaces for your EpiCenter datamart. Based on your input, this shell script produces an SQL script that contains the appropriate CREATE TABLESPACE commands for your datamart. If you have not already done so, follow the instruction in Step Step 2, on page 38 of the Configuring Oracle section to download this script.

Use the **epi_tsp.sh** shell script to specify the number of files to create for the following tablespaces:

- Large fact tables
- Indexes on large tables
- Dimension tables
- Indexes on dimension tables
- Metadata tables
- Transient tables
- Other application-specific tables

Take the following steps to use this script:

1. Log in as **oracle**, then enter the following command:

```
$ORACLE_HOME/rdbms/admin/epi_tsp.sh
```

The **epi_tsp.sh** script issues a series of prompts to guide you through the process of creating tablespaces for your datamart. These prompts indicate default values for sizes and file counts that are calculated based on the initial settings that you specify for the file size and number of files in the fact-table tablespace. In some cases, such as for the metadata tablespace, file sizes are fixed.

2. Edit the **epi_tsp.sql** script to ensure that the sizes and pathnames for datafiles are correct. EpiPhany recommends that you avoid reducing the file size for the metadata tablespace below 100 megabytes.
3. Enter the following shell command:

```
sqlplus internal
```

4. Enter the following **sqlplus** command:

```
@epi_tsp.sql
```

Note: *This SQL script creates tablespaces with specific names that EpiManager and EpiChannel recognize by default. If you use alternate names for tablespaces, you must define values for the macros that EpiChannel uses to locate those tablespaces. Refer to the Epiphany System Guide for details on E.piphany macros.*

CREATING EPIMART AND EPIMETA USERS

Epiphany provides a sample SQL script, called **epi_usr.sql**, that you can use to create standard users (owners) for EpiCenter datamart and metadata tables. You must edit this script to specify the passwords for the EPIMETA and EPIMART users. After you have done so, you can take the following steps to run this script:

1. Enter the following shell command:

```
sqlplus internal
```

2. Enter the following **sqlplus** command:

```
@epi_usr.sql  
exit
```

Warning: *You can perform this and other Oracle configuration task with the Security Manager utility of the Oracle client package for Windows NT Server. If you do use the client package to administer your Oracle database, do not open the Enterprise manager as user EPIMETA. Enterprise Manager creates spurious tables in the EPIMETA tablespace.*

This completes the configuration instructions for Oracle and the preparations required for the datamart host. For information on preparing your Epiphany application host, please proceed to the next section.

PREPARING YOUR APPLICATION HOST

E.piphany applications require a minimum of 64 megabytes of RAM on your application host. You must install the following software packages on your application host before you install E.piphany software:

- Windows NT Server 4.0
- Microsoft Internet Information Server (IIS), Version 4.0
- Selected components of Microsoft Office

If your application host is not the same computer as your datamart host you must also:

- Install the appropriate client software for the database server on which your datamart resides.
- Configure connectivity for your client software.

The following sections describe these activities.

INSTALLING WINDOWS NT SERVER 4.0 ON YOUR APPLICATION HOST

If you plan to install your E.piphany application on the same host as your datamart, you do not need to install Windows NT Server again. Otherwise, please see, "Windows NT Server 4.0 Installation," on page 18, and "Network Connectivity," on page 20 for instructions on installing and configuring this operating system.

INSTALLING INTERNET INFORMATION SERVER, VERSION 4.0

Epiphany requires that you install Microsoft Internet Information Server (IIS) 4.0. IIS 4.0 is distributed with Windows NT Server, Options Pack 4.0. Detailed instructions for installing components of this options pack are provided on the installation disk. You can install additional components of the options pack on your application host if you choose.

Follow these recommendations when setting up IIS:

- Use the default values for the Options dialog box.
- You may specify different values in the Database Publishing Directory dialog box. To use the security features of IIS, be sure that the **WWWroot** directory resides on an NTFS file system.
- Select the Microsoft SQL Server driver to install.
- Set the Time Zone.
- In the Windows NT Server Services dialog, set the following services to Manual start-up: Certificate Authority and Content Index.

After installing the package, please reboot Windows NT Server.

For further information about IIS 4.0, please refer to the Microsoft support site:

<http://www.microsoft.com/support>.

INSTALLING MICROSOFT OFFICE COMPONENTS

You can install Microsoft Office from the Microsoft Office CD. The installation program provides detailed instructions. Please follow those instructions to install the Microsoft Exchange mail client, ODBC connection client, and any other components of Microsoft Office that you choose to include.

INSTALLING AND CONFIGURING CONNECTIVITY PACKAGES FOR YOUR DATABASE CLIENT SOFTWARE

If you plan to install E.piphany software on the same computer as your datamart, you do not need to install or configure additional client software. You can instead turn to Chapter 2, "Installing E.piphany Software." If you plan to install your E.piphany application on a separate computer, please proceed with the sections that follow.

SQL UTILITIES

If you are using SQL Server for your datamart, you must install the SQL Utilities, option on the SQL Server installation CD, which provides appropriate instructions.

ORACLE8 UTILITIES

If your datamart resides on an Oracle database server, you must install the following utilities:

- Oracle8 Client for Windows NT Server
- Version 8.0.4.4.0 of the Oracle ODBC driver

Please take the following steps to install these packages:

1. In the Select Installation Options dialog, choose Oracle8 Client.
2. In the Select Oracle8 Client Configuration dialog, choose Database Administrator.
3. Install the Oracle8 Client package:

The Oracle installer installs Net8 as part of the client-utilities package. You use Net8 to establish connectivity with your datamart. To establish a connection to the Oracle instance on which the datamart resides, you must provide Net8 with the following information about that instance.

You can use the Net8 Easy Config utility that comes with Oracle8 Client, or you can update the **tnsnames.ora** file directly. Either way, you must supply the following information:

- The protocol, in this case TCP/IP
- The hostname or IP address of the datamart host
- The port number of the TNS listener service on the datamart host
- The Oracle instance name (SID)

The **tnsnames.ora** file is located in the **Orant\Net80\Admin** folder in your Oracle installation directory. A typical entry for an E.piphany datamart takes the following form:

```
ORCL=
  (DESCRIPTION=
    (ADDRESS=
      (PROTOCOL = TCP)
      (HOST = hostname)
      (Port = port)
    )
    (CONNECT_DATA = (SID = ORCL)
  )
)
```

Replace *hostname* with the name of the computer on which your Oracle database server resides. Replace *port* with the port number for your database server. For Oracle, the default port number is 1521. Replace each occurrence of *ORCL* with the actual SID of the instance on which your EpiCenter datamart resides.

You must ensure that the Net8 entry on the application host matches the Net8 entry for your EpiCenter instance on the datamart host. If necessary, you can create a Net8 alias for the EpiCenter instance on the datamart host. For additional details on configuring Net8, refer to the *Net8 Getting Started* manual.

4. Use the Oracle installer to install Version 8.0.4.4.0 of the Oracle ODBC driver, which you can download from the following Web site:

http://www.oracle.com/products/free_software/index.html

This completes the tasks that you must perform to prepare your E.piphany application host. Please turn to Chapter 2 for instructions on installing E.piphany software.

INSTALLING E.PIPHANY SOFTWARE

This chapter describes the procedures for installing and configuring E.piphany application software.

Warning: If you are upgrading from an earlier version of E.piphany software, you must export EpiMeta metadata from your existing E.piphany application before you install the current version of E.piphany software. Refer to "Exporting Metadata," on page 61 for details.

The standard E.piphany software installation includes:

- the EpiManager™ administrative utility for configuring and managing your datamart and applications.
- the AppServer™ application server, which coordinates user requests for data and reports, returns results in HTML format, and supports navigation between E.piphany Web pages.
- the EpiChannel™ extraction facility for downloading data from source systems into your EpiCenter datamart.
- other E.piphany components and utilities.
- drivers and other third-party software components on which E.piphany software depends.

Note: E.piphany application software does not have to be installed on the same computer on which your EpiCenter datamart resides.

The computer from which you run the E.piphany Application Server (AppServer™) must provide:

- Windows NT Server 4.0 set to run in 256-colors mode or higher
- 64 megabytes of RAM for use by E.piphany software

09625518 072500

- Microsoft Internet Information Server (IIS), Version 4.0

If IIS is not already installed on your application host, you must install it before you attempt to install E.piphany software. Please review the guidelines for installing IIS listed in “Installing Internet Information Server, Version 4.0,” on page 45.

- Selected components of Microsoft Office

If the Microsoft Exchange Messaging client and the ODBC connection client are not yet installed on your application host, you must install it. Please review the guidelines for installing these components described in “Installing Microsoft Office Components,” on page 46.

If your EpiCenter datamart resides on a Windows NT Server host, you can use the same computer for your datamart and your E.piphany application.

The E.piphany installer offers to install several third-party system components that are required to support your E.piphany applications if they are not already present on your system. These components include:

- ODBC and JDBC drivers
- Sun JRE
- Microsoft Internet Explorer 4.0 (IE), including the Microsoft Java virtual machine

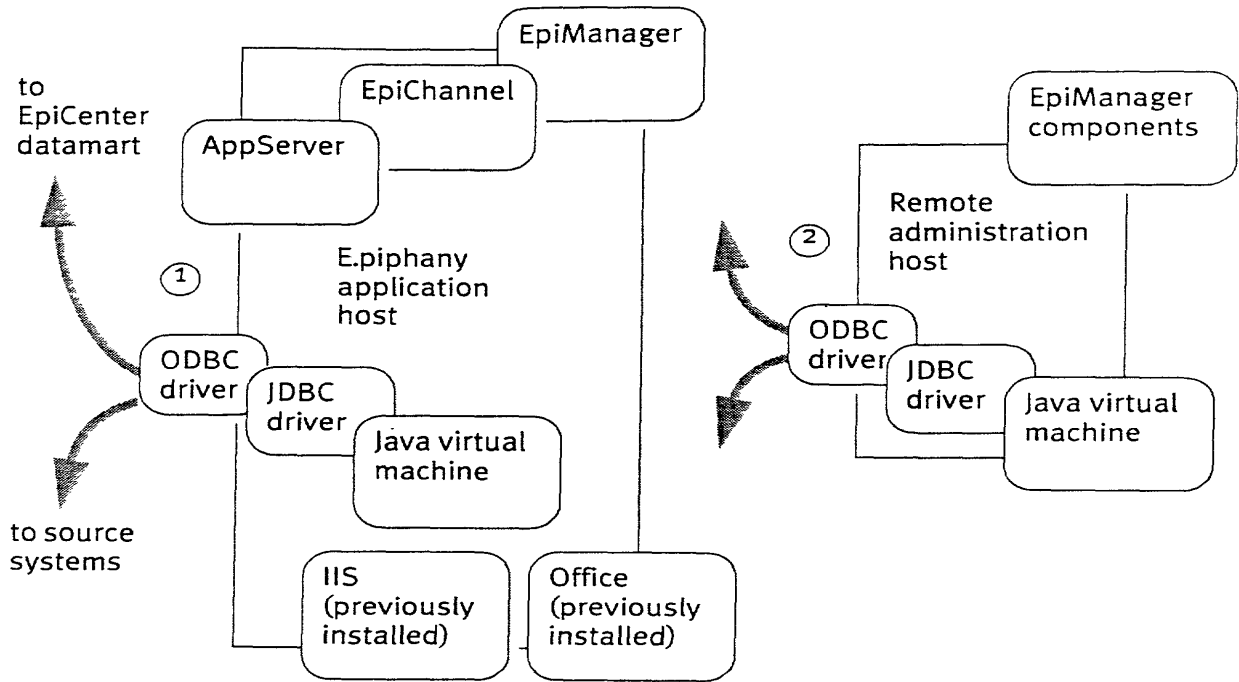
Please install IE 4.0 if the installer asks you to do so. The installer prompts you to exit if the Java virtual machine is not present and you do not install it.

After you have installed E.piphany software on the on the application host, you can install individual components of the EpiCenter Enterprise Manager (also referred to as EpiManager™) administrative utility on other computers to provide easier access for specific administrative tasks.

To install the complete set of standard E.piphany software on your application host, choose the Application Server installation option. To install an additional copy of selected E.piphany utilities on a remote host (which can run Windows 95, Windows 98, or Windows NT Server 4.0), choose the Remote Administration option.

Figure 2 illustrates the components that you install on the E.piphany application host and any remote-administration hosts that you might add.

FIGURE 2: EPIPHANY APPLICATION COMPONENTS



PERFORMING THE INSTALLATION

Take the following steps to install E.piphany software on your application host:

1. Log in to an account that has administrator privileges on the local host.
2. Exit all open Windows applications.

If an AppServer instance is running, from the Start menu, go to **Settings\Control Panel\Services** and manually stop the Application Server. If you do not, the installer cannot properly download the DLLs that this release requires.

3. Insert the Epiphany CD into your CD-ROM drive.
4. Double-click the **Setup.exe** icon in the CD-ROM folder.
5. If the Microsoft Java virtual machine is not installed on your system, you are requested to install it. Please choose Yes in response to this prompt.
6. The System Configuration screen displays your system's configuration data (for your information).

Follow the instructions below for either the Application Server or Remote Administration installation options. If you are installing E.piphany software for the first time, choose the Application Server option.

Warning: The installation program prompts you to reboot when the installation process is complete. If you answer Yes, you must again log in to a user account with administrator privileges to ensure that all new versions of .dll and other files are properly registered.

APPLICATION SERVER

The following steps apply to the Application Server installation option.

1. Choose the Application Server option.
A Services window shows which services will be stopped before the copy operation begins.
2. Enter the instance name. This is the name of the service as it will appear in the Service Control Panel. If you are upgrading from a previous version of E.piphany software, you must use a different instance name than that of your existing Epiphany instance.

An instance name distinguishes among multiple installations of the Epiphany software on the same machine. Typically, you will install the software once on a machine and thus have one instance. You are asked if this instance will be the default instance. If so, the **Setup** program makes this instance the default Web server destination.

3. Enter the machine name on which the Application Server runs.

By default, this is the full DNS name of the server. Because of your local network configuration, you may need to use a unique name: for example, the machine name without the domain name. Check with your local system network administrator if you have any questions.

4. Accept the default TCP/IP port for the Application Server if there is only one instance of the Application Server.

Each instance must have a separate TCP/IP port. Check with your local system network administrator if this is the case.

5. If your datamart resides on SQL Server, enter the name of the database server and the database name. The database name is the name of the EpiMeta database. This database does not need to exist as of yet. (The Registry keys are populated based on what you enter.)

If your datamart resides on an Oracle database, enter the hostname or IP address of the computer on which that database resides.

6. If your datamart is on SQL Server, enter the user name and password for the database server. The user must have system administrator (SA) privileges on the EpiCenter host.

If your datamart resides on Oracle, enter the user name for your EpiMeta user (typically **EPIMETA**) and the password for that user.

7. At this point, the installer asks if the instance name you entered in Step 1 is the default instance name. If you are installing Epiphany software for the first time, or if you plan to run a single instance of AppServer on this host, click Yes.

If you click No, your Web Server will not be set up to automatically route users to the Epiphany login page. In that case, users can access the login page by entering:

`http://machine_name/scripts/instance_name/Epiphany.dll`

8. Select the destination directory for the Epiphany software. The default directory is:

`C:\Program Files\Epiphany\instance_name`

9. Enter the location of the run-time reports and charts that end users will access. The default location is:

`C:\Program Files\Epiphany\instance_name\Charts`

10. Create a new folder as requested.

11. Select Administrative Tools from the components list. If you want to install Epiphany documentation on your hard disk, choose Documentation also.

- Administrative Tools

If you are installing E.piphany software for the first time, choose the components that have been selected by default.

- Documentation

E.piphany provides technical manuals on line in PDF format with keyword-search indexes. To use the PDF full-text index, you must have Acrobat Reader with Search or Acrobat Exchange. You can download Acrobat Reader 3.0 from the following Web site:

<http://www.adobe.com/prodindex/acrobat/readstep.html>

12. Select the default program-icon location.

13. If you are performing an upgrade from a previous E.piphany release and your datamart is already populated with data, please refer to Chapter 3, "Preparing to Migrate From Previous Releases," for further instructions.

This completes the Application Server installation option.

09625518.072500

REMOTE ADMINISTRATION

1. Select the destination directory for the Epiphany software. The default is **C:\Program Files\Epiphany\instance_name**.
2. Enter the location of the run-time reports and charts that end users will access. **C:\Program Files\Epiphany\instance_name\Charts** is the default. Create a new folder as requested.
3. Select the Administrative Tools from the list of components to be installed: If you want to install E.piphany documentation on your hard disk, choose Documentation also.

- Administrative Tools

You have the option of selecting components of the following Administrative Tools: EpiCenter Manager, Web Builder, and Security Manager. Although EpiCenter Manager includes Web Builder and Security Manager, these components can be installed separately.

A person who does not need to have access to all of the features of EpiCenter Manager can use one of these components. The front-end configuration features of EpiCenter Manager are available through Web Builder. Access rights and system permissions can be administered using Security Manager.

- Documentation

E.piphany provides technical manuals on line in PDF format with keyword-search indexes. To use the PDF full-text index, you must have Version 3.0 or higher of Acrobat Exchange or Acrobat Reader. You can download Acrobat Reader 3.0 from the Adobe Web site:

4. Select the default program-icon location.
5. If you are performing an upgrade from a previous E.piphany release and your datamart is already populated with data, please refer to Chapter 3, "Preparing to Migrate From Previous Releases," for further instructions.

This completes the Remote Administration software-installation option.

CONFIGURING E.PIPHANY SOFTWARE

After you have installed E.piphany software, you must configure the E.piphany Web-server proxy. In addition to configuring the proxy, E.piphany also recommends that you enable performance monitoring for data-extraction operations.

CONFIGURING THE E.PIPHANY WEB-SERVER PROXY

The Epiphany Web-server proxy interacts with IIS 4.0. For the Epiphany Application Suite to function properly, set the IIS configuration values according to the instructions in this section.

1. Open the IIS 4.0 Service Manager from the Start menu by choosing Programs, then Windows NT Server 4.0 Option Pack, then Microsoft Internet Information Server, and then Internet Service Manager.
2. Right-click your server icon and select Properties from the pop-up menu. The Properties dialog box is displayed in the Master Properties panel. Select WWW Services as the Master Properties. Click Edit. In the WWW Service Master Properties dialog box, select Directory Security. In the Anonymous Access and Authentication Control Panel, click Edit. The Authentication Methods dialog box is displayed. Select Allow Anonymous Access. Click Edit.
3. Configure Anonymous Login with file access permissions for reading and writing to all Epiphany files. Enter the following attributes for the Anonymous user account:
 - Username
A username that has file access permission for all of the Epiphany installed files and directories. This username also needs permission to read the Epiphany entries in the Registry.
 - Password
The password for this username.

- Make sure that the directories that contain the **Epiphany.dll** and the **makechart.dll** files have execute permissions.

SETTING UP NT PERFORMANCE MONITORING FOR EXTRACTION

The **extract.exe** program is instrumented to use the standard NT Performance Monitoring facility, but you must take the following steps to enable the display of Epiphany data extraction processes in the NT Performance Monitor:

1. Your **Epiphany\instance_name\win32** installation directory must contain these items: **EpiPerfMon.dll**, **EpiPerfMon.ini**, and **EpiPerfMon.reg**. **EpiPerfMon.reg** has values specified for the following Registry key:

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\
instance\Performance
```

Enable write permission for the **EpiPerfMon.reg** file, then edit this file to ensure that the value of **Library** points to the correct folder in which **EpiPerfMon.dll** resides; that is, your current Win32 directory. Change the **instance_name** to the correct value for the **Library** path in this file.

2. Run **EpiPerfMon.reg**. To check the results, open the Registry and navigate to the key specified in Step 2. Ensure that the path supplied for **Library** is correct. If it is not, the Epiphany Channels object entry does not appear in the object list for the Performance Monitor.
3. From a command line, go to the **Win32** directory of the current instance and run the following commands to unload parameters from any previous Registry entry and initialize the values associated with your Epiphany application:

```
unlodctr Epi
lodctr EpiPerfmon.ini
```

Note: **Epi** is the default driver and key specified in **EpiPerfmon.reg**.

You can now start the NT Performance Monitor and monitor the progress of data-extraction jobs performed by EpiChannel.

VERIFYING YOUR INSTALLATION

When you install the Epiphany software, the first screen displays your system configuration.

After installing the Epiphany software you can:

- Run the E.piphany tools, such as EpiManager, to configure your EpiCenter and set up EpiChannel extraction jobs. You will also use EpiCenter to construct the topics and Web pages that enable users to query the data in your data warehouse.

The *Epiphany System Guide* gives instruction on how to use EpiManager to administer your E.piphany application. Please read Chapters 1 and 2 of that guide for background information before using EpiManager.

- Enter your e-mail password to receive notification of Epiphany system status. You will use the Configuration dialog box in EpiManager to set this up. Instructions for doing so appear in Chapter 3 of that guide.
- Run the E.piphany extraction program (EpiChannel) to extract data from your source systems and place them in your EpiCenter datamart. EpiChannel performs the extraction jobs that you defined in EpiCenter Manager.
- Start AppServer, the Epiphany Application Server. Refer to the *Epiphany System Guide* for details.

09625518.072500

PREPARING TO MIGRATE FROM PREVIOUS RELEASES

This chapter describes the preparations you must make before you can upgrade from a previous E.piphany release to Release 4.0. These preparations include:

- Migrating an EpiMart database that resides on SQL Server to SQL Server Version 7.0
- Migrating EpiMart database that reside on Oracle to Oracle8 Version 8.0.5.1.0
- Exporting EpiMeta metadata from your earlier E.piphany release for later import into your Release 4.0 EpiMeta database
- Modifying extraction jobs to eliminate obsolete data types

This chapter describes each of these actions. Chapter 3 of the *E.piphany System Guide* describes the steps that you take to complete the upgrade process after you have installed E.piphany e.4 Release 4.0.

MIGRATING AN EPIMART DATABASE

The sections that follow describe the preparations you must take to migrate your datamart to a database server that E.piphany e.4 Release 4.0 supports.

SQL SERVER

If you are upgrading your E.piphany application from Release 3.3 or earlier, and you plan to continue using SQL Server as the database server for your datamart, you must:

- Install and configure Version 7.0 of SQL Server if you have not already done so. Refer to Chapter 1, “Preparing to Install E.piphany Software,” for details.
- Truncate your existing staging tables to eliminate the possibility of null-value errors that could halt the process of upgrading your datamart. These staging tables contain temporary data only. Truncating them has no lasting effect on your datamart. Take the following steps:

1. Using the **isql** utility, connect as user **dba** to the database that stores your EpiMeta metadata, then enter the following SQL commands:

```
select 'truncate table ' + stage_name
      from dim_base_view
      where dim_base_type = 'Default'

select 'truncate table ' + stage_name
      from fact_tbl_view

select 'truncate table ' + external_tbl_name
      from external_tbl
```

2. Save the output of the commands you ran in Step 1. This output takes the form of a set of SQL statements.
 3. Connect to the database that stores your EpiMart datamart data, then enter the output that you saved in Step 2 as commands in **isql**. These commands truncate the staging tables when run against the datamart.
- Follow the procedure provided by the SQL Server 7.0 installer to upgrade your EpiMart database.

ORACLE

If you are upgrading your E.piphany application from Release 3.4 or earlier, and you plan to continue using Oracle as the database server for your datamart, you must:

- Install and configure Version 8.0.5.1 if you have not already done so. Refer to Chapter 1, “Preparing to Install E.piphany Software.” for details.
- Follow the procedures described in the Oracle8 Installation Guide to upgrade your EpiMart database.

EXPORTING METADATA

This section describes the steps that you must take to preserve existing metadata prior to installing the current version of E.piphany software.

EpiManager provides a facility for upgrading your existing installation. Rather than operating on the EpiMeta database itself, the upgrade procedure uses a standard E.piphany export file in Microsoft Access format. You use this facility to export existing metadata from your older version of E.piphany software *before* you install your new version of E.piphany e.4 software. After you have installed the current version of E.piphany software, you can import the metadata from the export file and perform additional steps to optimize it for use with in current release

Take the following steps to export your existing metadata:

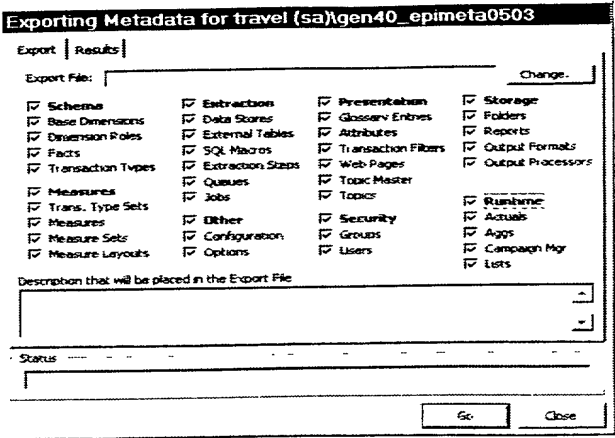
1. Use the **EpiCenter/Export/Export All** command in the EpiCenter Manager menu to generate a full export file of your existing metadata.

By default, the Actuals and Aggs are not exported when you issue the Export All command. If you intend to preserve an existing datamart, select *both* of these options in the run-time section of the Exporting Metadata dialog. (See Figure 3, on page 62.) If you plan to create a new datamart, you do not need include these two options.

2. Use the **EpiManager\EpiCenter\Initialize EpiCenter** command in EpiManager to build a new EpiMeta database.

09625518.072500

FIGURE 3: EXPORTING METADATA DIALOG BOX



Refer to Chapter 3 of the *Epiphany System Guide* for a complete description of the procedure that you follow to migrate the metadata and saved reports that you have preserved in the export file.

MODIFYING EXTRACTION JOBS

This release of E.piphany software no longer supports the following data types:

- NUMBER (9)
- NUMBER (9 , 2)
- FLOAT

You must replace all references to these data types in extraction jobs with references to one of the following new physical data types before you can extract data into an E.piphany e.4 Release 4.0 datamart:

- EPIINT
- FACTMONEY
- FACTQTY

09625518.072500

0-0-E-TP-EIG-4.0





E.PIPHANY E.4 SYSTEM GUIDE

RELEASE 4.0

JUNE 1999

Epiphany Confidential

E.PIPHANY E.4 SYSTEM GUIDE

RELEASE 4.0

JUNE 1999

Epiphany Confidential

09625518.072500

09625518.072500

COPYRIGHT AND TRADEMARKS

Copyright © 1997-1999 by E.piphany, Inc. All rights reserved.

This document and the software it describes are furnished under license and may be used or copied only in accordance with such license. Except as permitted by such license, the contents of this document may not be disclosed to third parties, copied, or duplicated in any form, in whole or in part, without the prior written permission of E.piphany, Inc.

This document contains propriety and confidential information of E.piphany, Inc. The contents of this document are for informational use only, and the contents are subject to change without notice. E.piphany, Inc. assumes no responsibility or liability for any errors or inaccuracies that may appear in this document.

Unpublished rights reserved under the copyright Laws of the United States.

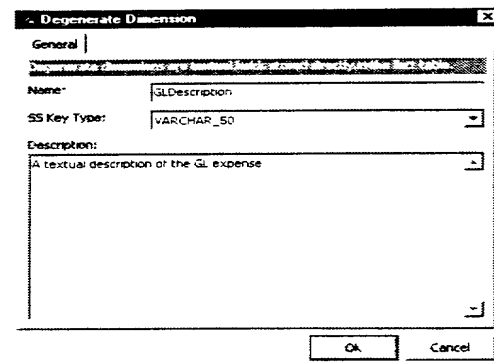
E.piphany™ and E.piphany e.4™ are trademarks of E.piphany, Inc. All other products or name brands are trademarks of their respective holders.

Printed in the USA

RESTRICTED RIGHTS LEGEND

Software and accompanying materials acquired with United States Federal Government funds or intended for use within or for any United States federal agency are provided with "Restricted Rights" as defined in DFARS 252.227-7013(c)(1)(ii) or FAR 52.227-19.

FIGURE 44: DEGENERATE DIMENSION DIALOG BOX



FACT TABLES

A fact table is a physical database table that holds numeric data in its columns. It also represents the intersection of a series of dimension keys. A fact table consists of dimension role foreign keys, degenerate dimension keys, and fact columns.

An EpiCenter that utilizes Campaign Manager has additional built-in fact tables: **Ind_Group_Joiner**, **Communication**, and **SeedJoiner**. The **Ind_Group_Joiner** table specifies the association between members of the base dimensions that correspond to the `indiv` and `group` dimension roles. Each individual must belong to exactly one group. If you want to turn off the processing of individuals (for a group-only EpiCenter), you can remove the `indiv` dimension role from the **Ind_Group_Joiner** table.

The **Communication** table is the special backfeed fact table for campaigns that shows who was sent a particular mailer during a campaign. A typical row of the **Communication** fact table contains information about a specific treatment (promotion) applied to a single individual. This fact table captures information related to the entire campaign's communication history (current and previous marketing programs) and makes it available to the datamart for query.

The **SeedJoiner** fact table is where the seed data resides. This fact table must be present in order for seeding to occur.

TRANSACTION TYPE AND DATE DIMENSIONS

When defining fact tables, remember that the transtype (transaction type) and date dimensions occur in every fact table.

The transtype is a “slice” of the fact table that functions as a separate fact table. You can configure one fact table with multiple transtypes, or configure multiple fact tables. The advantage of having multiple transtypes is that measures that require two transaction types, such as BOOK and BOOK_RETURN need to issue only one query. The advantages of multiple fact tables are that queries that need only one transtype have fewer rows to navigate, and that aggregate tables are smaller.

Because the date dimension can serve as the base dimension of any dimension role, a fact table can receive more than one foreign key to the date dimension (all fact tables receive `date_key` as a foreign key). You can also define additional date dimension roles and include them in the fact table. The column name in the fact table is *dimrolename_key* (note that the suffix is not *sskey*).

Note: Because of the special usage of date_key, it cannot be used as an attribute; use day_name as an alternative.

As with all dates in the system, only the day-level granularity can be used for these dimension roles. The time should be 12:00 midnight during extraction, which can be accomplished by using built-in macros, such as `$$TO_EPIDATE`.

Fact semantics treat these columns in the usual way. Changes to the dimension-role value result in the booking or debooking of those fact tables that allow the changing of facts. All columns in the date dimension are available for querying. The AppServer sorts these columns in the same manner as the built-in date dimension.

DEFINING A FACT TABLE

To define a fact table:

1. Right-click the **Facts** folder icon, and select **New Fact**. The Fact Table dialog box (Figure 45) is displayed. This dialog box has tabs labeled General, Dimensionality, Indexes, List Manager, Aggregate Instructions, and Built Aggregates.
2. In the upper pane of the General tab, enter the fact table's name. The maximum length is 20 characters.
3. Select the source system key (**sskey**) data type from the drop-down list.
4. Enter a textual description of this fact table for your reference.

FIGURE 45: FACT TABLE DIALOG BOX: GENERAL TAB

Fact: ChanInvoices

General | Dimensionality | Indexes | List Manager | Aggregate Instructions | Built Aggregates

Fact Name: ChanInvoices

SSKey Type: VARCHAR_50

Description:

Fact Columns:

Column Name	Amount	Units

Add Column Edit Remove

OK Cancel Help

DEFINING FACT COLUMNS

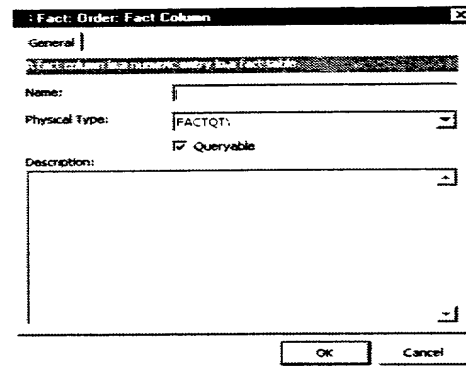
Fact columns contain the actual customer numeric data, such as `net_price` or `number_units`.

You can use the **Fact Columns** pane of the General tab of the Fact Table dialog box to define a fact column.

Follow these steps to define a fact column:

1. Click **Add Column**. The Fact Column dialog box (Figure 46) is displayed.

FIGURE 46: FACT COLUMN DIALOG BOX



2. Enter the column name and select the physical type from the drop-down list. The physical type you select is replaced by its associated database type. The translation of physical type to database type depends on your RDBMS platform. See Appendix D, “Physical Type Values” for descriptions of these physical types.

Note: For this release, the **FACTMONEY**, **FACTQTY**, and **EPIINT** physical types are supported for fact columns.

3. The **Queryable** option is checked by default. If this option is not checked, the column cannot be moved into List Manager-related tables.

List Manager creates composite indexes on fact tables to improve performance.

These indexes can have no more than 16 columns in SQL Server 7.0. Exceeding this number results in degraded performance. Note the following SQL Server 7.0 restriction:

$$16 \geq 2 * (ind_d_roles) + grp_d_roles + qry_dims + qry_fcts$$

Replace *ind_d_roles* with the number of individual dimension roles.

Replace *grp_d_roles* with the number of group dimension roles. Replace *qry_dims* with the number of dimensions with at least one queryable attribute. Replace *qry_fcts* with the number of queryable fact columns.

4. Enter a description, and click **OK** to return to the Fact Table dialog box. The new fact column appears in the list box.
5. Repeat the above steps to define another column.

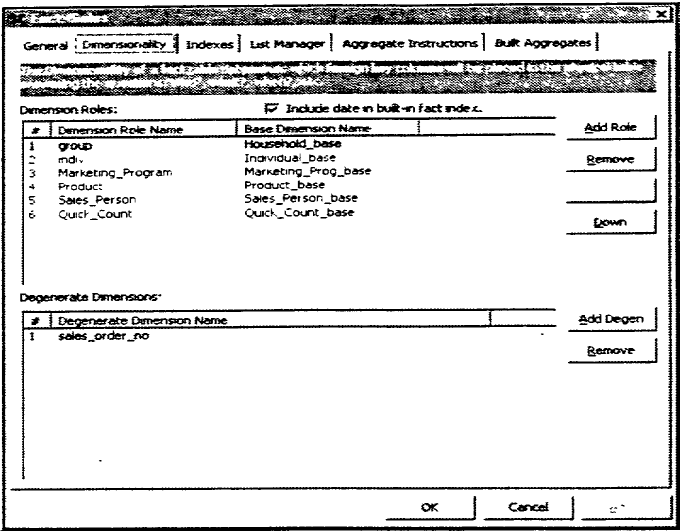
To remove a column from the fact table, select it and click **Remove**. To modify the column description, select it and click **Edit**.

FOREIGN KEYS IN FACT TABLES

Foreign keys in fact tables reference dimension tables. In a star schema, the fact table is in the center of the star, and the points that radiate outward are the dimensions (attributes stored in dimension tables). Fact tables store keys that reference these “foreign” dimension tables. There are two kinds of foreign keys: dimension roles and degenerate dimensions.

You can use the Dimensionality tab of the Fact Table dialog box (Figure 47) to assign up to 20 keys to a fact table.

FIGURE 47: FACT TABLE DIALOG BOX: DIMENSIONALITY TAB



To assign a dimension role foreign key:

1. In the **Dimension Roles** pane, click **Add Role** and select one or more dimension roles from the Choose dialog box.
Clicking the **New** button in the Choose dialog box opens the Dimension Role dialog box for creating a new dimension role.
2. Add all appropriate dimension roles.
3. Use the **Up** and **Down** buttons to order the dimension roles for the default index. Only the first dimension role (the leading term) is of significance.

By default, each fact table has a single clustered index with the leading term of `date_key` which allows fast filtering based on time. (The leading term of an index is the most important term; for example, a phone book is indexed by last name; you cannot locate someone by first name only.)

The built-in fact tables **Communication** and **Ind_Group_Joiner** have the dimension roles **indiv** and **group** that you need to associate with the base dimension tables that contain individual and group data. Initially, these roles are set to *campaign* and *cell*, respectively.

The built-in **SeedJoiner** fact table has the dimension roles **indiv** and **group** that are already associated with the base dimension tables that contain individual (**IndivSeed**) and group (**GroupSeed**) data.

The absence of **indiv** in the **Ind_Group_Joiner** or **SeedJoiner** fact table indicates that the EpiCenter is only for a group. Thus for a group-only EpiCenter, remove the **indiv** dimension role for the **Ind_Group_Joiner** and **SeedJoiner** fact tables. The **indiv** dimension role itself, however, cannot be removed. It must always point to some base dimension table, even when it is not used.

To assign a degenerate dimension foreign key:

1. In the **Degenerate Dimensions** pane, click **Add Degen** and select one or more degenerate dimensions from the Choose dialog box.

Clicking the **New** button in the Choose dialog box opens the Degenerate Dimension dialog box for creating a new degenerate dimension.

2. Add all appropriate degenerate dimensions.

By default, fact tables have the leading term of the index as **date_key** because a wide class of queries filters on the time dimension. This indexing allows for fast filtering based on time. You have the option of removing the **date_key** from the index. To remove it, uncheck **Include date in built-in fact index** in the Dimensionality tab of the Fact Table dialog box (Figure 47, on page 196).

If end users typically apply selective filters to other dimension roles such as customer or product, then configuring custom indexes with these leading terms improves system performance. (See “Custom Fact Indexing,” on page 92 for a more in-depth discussion of this topic.)

CUSTOM FACT INDEXES

In general, aggregates satisfy high-level queries while indexes satisfy highly selective queries. Query drill-downs on a Web page transition from broad to selective. In terms of aggregates and indexing, the sequence of a drill-down transition from top to bottom might be—

1. A small aggregate (for example, Business Unit equals Copiers).
2. A larger aggregate (month equals January 1999).
3. An index on a large aggregate (Customer Region equals West).
4. An index on a base dimension table (drill down by Customer Name equals John Doe).

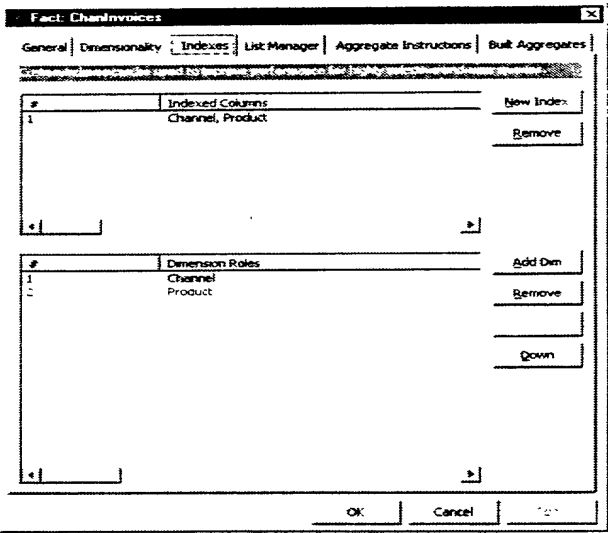
Note: Aggregate tables are indexed in the same way as their base tables on those columns that still exist in the aggregate. Aggbuilder does not build the same index twice simply because of missing columns.

A custom fact index is the metadata definition of an index to build on a fact table in EpiMart. It is an ordered list of dimension roles. You can use EpiCenter Manager to configure custom indexes to improve query performance significantly for selective queries.

You can use the Indexes tab of the Fact Table dialog box (Figure 48, on page 199) to create a new custom index:

1. In the **Indexed Columns** pane, click **New Index**.
2. Select one or more dimension roles from the **Choose** dialog box to add to this index. The first dimension role is the leading term.

FIGURE 48: FACT TABLE DIALOG BOX: INDEXES TAB



To modify dimension roles for an index: In the **Dimension Roles** pane, select the index and click **Add Dim** to add additional roles.

To delete the index, select it and click **Remove**.

To remove dimension roles from an index, select the role in the lower pane, and click **Remove**.

THE LIST MANAGER TAB: CLUSTERS AND COUNTS

Defining clusters for fact tables can improve performance for List Manager queries. A *fact cluster* is an index that you set up to include a significant attribute (a leading dimension role) that end users typically query for a fact table. The E.piphany system generates additional attributes for the index. For example, assume an Order fact table in your star schema has dimensions for Customers, Product Lines, Sales Regions, and Sales Force. Users typically ask about buying patterns by Customers (the demographic dimension that you have associated with the group dimension role) and product lines. If you defined a cluster on the dimension role that points to Product Lines, then the query machinery can bypass unrelated dimensionality, such as Sales Force, for the Order fact table.

Note: When the Ind_Group_Joiner table serves as the fact table for transaction filters, you can define counts and clusters on it.

A *cluster* uses the SQL Server Covered Query or the Oracle Index FFS. These queries check if the attributes for a referenced fact table are part of an index, or cluster. If so, the search engine searches that cluster instead of all of the dimension tables that the fact table points to.

Designing meaningful clusters improves performance because all of the data for the query is contained within the index itself. Therefore, only the index pages, not the actual tables, must be referenced to retrieve the data.

Counts keep statistics about how often a row appears in the fact table. Counts are used by the List Manager query engine to select the best cluster for the fact table.

Note: Counts are used on Oracle databases to make queries run faster. Counts defined for SQL Server are ignored.

Guidelines for counts and clusters:

- You need to specify at least one cluster on a fact table for it to be available to the List Manager tables (using the List Manager tab of the Fact Table dialog box).

- To be able to cluster a fact table, you need to specify a dimension role that corresponds to a dimension other than a demographic dimension.
- *Oracle only:* Counts are associated with transaction types. Create a separate count for each transaction type associated with the fact table for this cluster. *Oracle does not use a cluster unless there is an associated count for the transaction type being queried.*
- Because cluster usage has some system overhead, whereas counts do not, be judicious in your use of clusters.

The List Manager tab of the Fact Table dialog box allows you to define clusters and counts.

FIGURE 49: FACT TABLE DIALOG BOX: LIST MANAGER TAB

Follow these steps to define a Cluster:

1. In the **Clusters** pane, click **Add Cluster**.
2. Choose a non-demographic dimension role for this cluster from the **Choose** dialog box. This dimension role should be the leading term.

(Oracle only) To define a count for this cluster:

1. In the **Counts** pane of the List Manager tab, click **Add Count**.
2. Select one of the associated transaction types for this fact table from the drop-down list. Click **OK**.
3. Choose the dimension role for the count from the Choose dialog box.
4. Repeat these steps to create a separate count for each transaction type associated with this fact table and cluster.

DEFINING FACT AGGREGATES

The Aggregate Instructions tab of the Fact Table dialog box (Figure 50) allows you to define fact table aggregation. The dimensions you choose to aggregate depend on which of the fact's dimensions end users might typically query as a group. EpiCenter Manager uses your specifications to generate the actual instructions in metadata that control how Aggbuilder builds aggregates for the fact table.

Having the system aggregate fact values for these dimensions in advance speeds up front-end query time. In general, when designing fact aggregates, attempt to strike a balance between broad fact aggregates (some compression/high query coverage) and narrow fact aggregates (high compression/low coverage).

To configure instructions for the fact table, open the Aggregate Instructions tab (Figure 50, on page 203).

FIGURE 50: FACT TABLE DIALOG BOX: AGGREGATE INSTRUCTIONS TAB

General | Dimensionality | Indexes | List Manager | **Aggregate Instructions** | Bulk Aggregates

Aggregate Groups:

Group Name	Dimension Roles
Aggreg	date, group, indiv, Marketing_Program, Product, Sales_P
Default	date, group, indiv, Marketing_Program, Product, Sales_P
indiv1	date, group, indiv, Marketing_Program, Product, Sales_P
indiv2	date, group, indiv, Marketing_Program, Product, Sales_P
nuAgg	date, group, indiv, Marketing_Program, Product, Sales_P
nuGroup	group, indiv, Product, transvpe

Buttons: Add Group, Edit, Remove, Duplicate, Generate

Actual Instructions: (Click to Sort) Enabled / Disabled: 0 / 0 Add Agg

On	Definition
----	------------

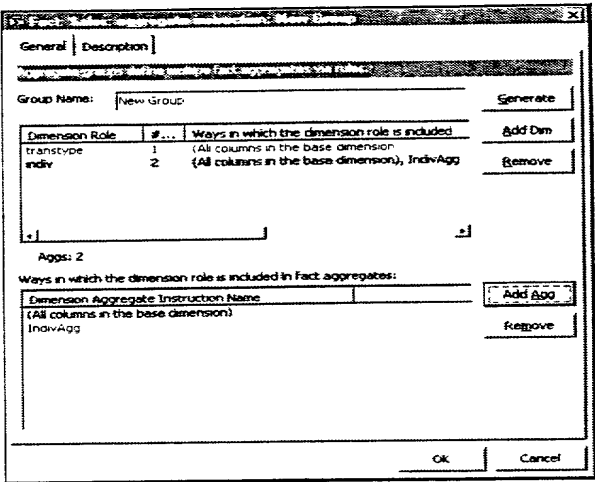
Buttons: Add, Edit, Remove, Duplicate, Generate

OK Cancel

Follow these steps to create aggregate groups:

1. Click **Add Group**.
2. In the General tab of the New Group dialog box (Figure 51, on page 204), enter the name of the group.
3. You can open the Description tab at any time to enter a description of this group.

FIGURE 51: NEW GROUP DIALOG BOX



4. The fact aggregates that can be generated from an aggregate group are determined by a combinatorial expansion based on the dimension role possibilities that you set up. Click **Add Dim** and select a dimension role for this group from the Choose dialog box.
5. All of a selected dimension role's base table columns display in the lower pane of the dialog box, in addition to the inclusive **All columns in the base dimension**. You can specify which columns are included in the aggregate by deleting those you do not want included. Select them in the lower pane and click **Remove**.
6. Add additional dimension roles for the group and specify the columns to be included.

The total number of aggregates for the group is shown on the tab.

7. Click OK after you have defined the new group. The group is added to the listing in the upper pane of the Aggregate Instructions tab.
8. Create additional groups as described above. To add a similar group, select it and click the **Duplicate** button. Click **Edit** to modify the group.

09625518 072500

After you have defined the groups, you can generate the actual instructions in metadata by clicking **Generate** in the Aggregate Instructions tab. Generating transfers the aggregate instructions to the **Actual Instructions** shown in the lower pane (Figure 50).

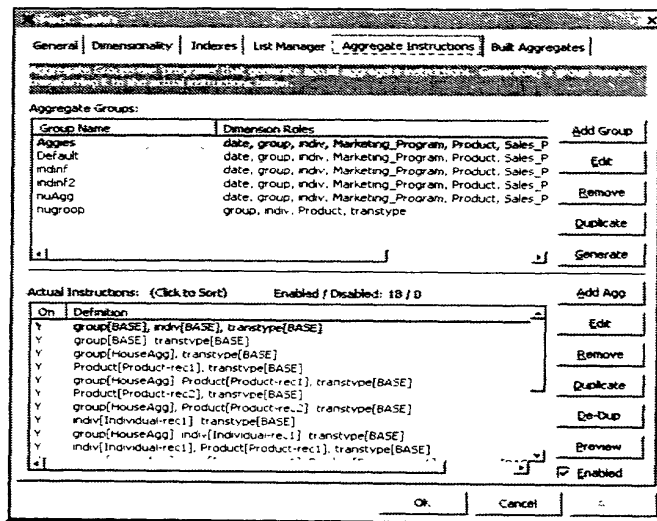
After you have added all of the instructions, click **De-Dup** to remove any duplicates that may have been created.

To sort instructions by dimension roles, click an item in the listing.

To disable aggregation for this fact table, uncheck **Enabled**.

Clicking the **Preview** button in the Aggregate Instructions tab opens the Aggregate Optimizer dialog box (Figure 36, on page 179).

FIGURE 52: AGGREGATE INSTRUCTIONS GENERATED



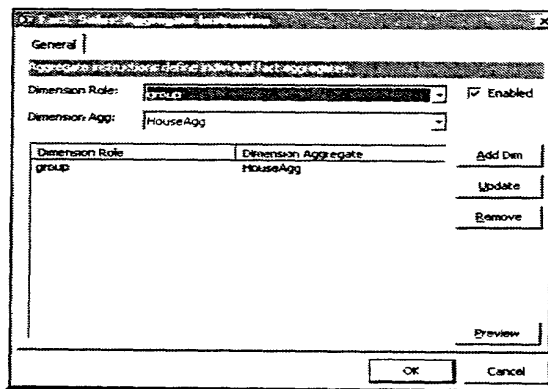
DEFINING/EDITING AGGREGATE INSTRUCTIONS

You can define an aggregate instruction by clicking **Add Agg** in either the **Aggregate Instructions** pane of the Aggregate Instructions tab, or the New Group dialog box. In the **Aggregate Instruction** dialog box (Figure 53) that appears, specify the dimension aggregate. Follow these steps:

1. Select the dimension role from the drop-down list.
2. Select the dimension aggregate.
3. Add any other dimension roles/dimension aggregates.
4. Click **OK** to add this aggregate instruction to the list of other defined instructions.

You can edit an aggregate instruction by double-clicking it, or by selecting it and clicking **Edit**, which displays the **Aggregate Instruction** dialog box (Figure 50).

FIGURE 53: AGGREGATE INSTRUCTION DIALOG BOX



REMOVING/DISABLING AGGREGATE INSTRUCTIONS

An aggregate group often produces some instructions that are infrequently used in actual user queries. Removing or disabling these can significantly reduce extraction time and disk usage.

If an aggregate instruction is not longer needed, you should disable it, instead of removing it. Removing the instruction in this one case might not remove it completely from the system, whereas disabling turns off all occurrences of the instruction. To disable an aggregate instruction, open the instruction's Aggregate Instruction dialog box and uncheck **Enabled**.

BROWSING FACT AGGREGATES

As part of refining your EpiCenter, you may wish to browse the list of fact aggregates that Aggbuilder built according to your instructions to determine which tables are available for the query machinery. You can use the Built Aggregates tab of the Fact Table dialog box (Figure 54) for this purpose.

You can browse the A/B/X/Y sets of tables as appropriate. Select the set from the drop-down list under the **Filter** button. There are four copies of every fact table, with suffixes **_A**, **_B**, **_X**, and **_Y**. The **_A** and **_B** tables are mirrored copies of a table containing recent facts, and their operation is similar to that of the **_A** and **_B** dimension tables. The **_X** and **_Y** tables are mirrored copies of a table containing historical (or less-recent) facts. At any time, either the **X** or **Y** history table is active. If the active tables are set to **A** and **X**, then all end-user queries that refer to a fact table are run against the union of the versions of that fact table that have **A** and **X** suffixes. The **B** and **Y** tables contain data that is one extraction older than the **A** and **X** tables, and they are inactive. The current datamart is shown in the lower-right corner of the EpiCenter Manager window.

You can also see which fact aggregates were *not* built for selected dimension roles. To do so:

1. Select **Not Included** from the **Dimension Aggregate** drop-down list.
2. Select a dimension role in the upper pane, and click **Filter**.
3. View the listing in the lower pane.

The **Fact Aggregate** pane shows the following information:

- The total number of fact aggregates built (with the current filter applied) is shown in the **Count** box.

- The fact aggregate number.

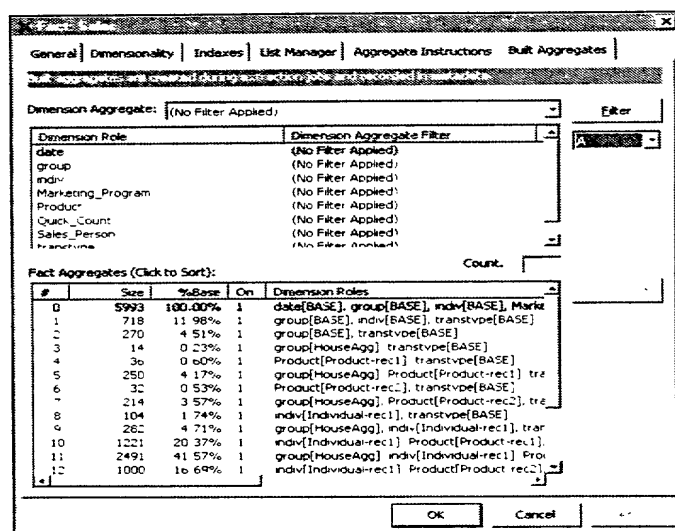
Although this numbering is arbitrary, it may be helpful when debugging query logs since the logs use these numbers.

- The **Size** and **%Base** (number of rows in this aggregate divided by the number of rows in the base fact table expressed as a percentage).

You can click the **Size** or **%Base** headings to sort aggregates by size. This sorting shows which aggregates are most valuable (a larger percentage means greater aggregation).

- The associated dimension roles.

FIGURE 54: FACT TABLE DIALOG BOX: BUILT AGGREGATES TAB



TRANSACTION TYPE SETS

Transaction types, such as Booked and Booked Returned, are defined in the Transaction Types tab (Figure 114, on page 420) of your Configuration dialog box. You are given a group of generic transaction types, and can use this dialog box to define other transaction types.

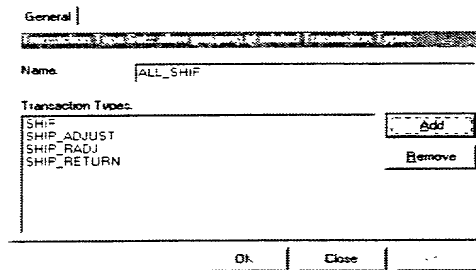
These transaction types can be grouped in a set and applied to a measure term. Measure terms, which are discussed in “Measure Terms,” on page 248, define the steps that determine how a measure that an end user selects on a Web page is calculated. A *measure term* is one component of an arithmetic expression that makes up a measure. It refers to the aggregation of a single fact column in a fact table, such as `SUM(Order.net_price)` with a transaction type, or a set of transaction types.

Because measure terms created via EpiCenter Manager specify a transaction type in addition to a fact column, only those rows in a fact table with the same transaction type can be summed. Applying a set of transaction types instead of individual transaction types enables the use of a single measure term calculation (usually `SUM`) to add up all rows for all transaction types in the set. You can use the Transaction Type Sets dialog box to group transactions into sets.

To define a transaction type set:

1. Right-click the **Transaction Type Set** folder icon and select **New Transaction Type Set**.
2. Enter the name of the set in the Transaction Type Set dialog box (Figure 55, on page 210).
3. Click **Add**. Select the transaction types in the set from the Choose dialog box.

FIGURE 55: TRANSACTION TYPE SET DIALOG BOX



This completes the instructions for defining your star schema. The next section, Extraction, describes how to configure the metadata for jobs that move the source data into these schema tables.

EXTRACTION

After you have configured your EpiCenter, you may use the **Extraction** folder to set up the extraction jobs. The **Extraction** folder consists of the sub-folders **Data Stores**, **Jobs**, **External Tables**, **Macros**, and **Queues**, each of which is discussed below.

DATA STORES

A data store is a logical location of data to be used either as a source (input data store) or sink (output data store) within an EpiCenter. Typically, the input data store is a source system database or file, and the output data store is EpiMart, which holds customer data, on an RDBMS server.

Epiphany provides two special default datamart stores: Epimeta and Epimart. Epimart is the typical datamart store. There are two default log stores: LoggingDB and JobFile Log. Associated with data stores are the default data store roles: Input, Logging, Output, and WorkingDir.

You can use the Data Store dialog boxes to create as many data stores as are appropriate for your site. The default data stores and any data store you create using the Data Store dialog box are available for selection in the **Object Gallery** when you define jobs.

The job dialog box is where you assign data stores and roles for an entire job. You can also assign data stores to individual job steps. For example, an extractor job has a single input and output data store. Because jobs consist of multiple extractors, a job has multiple input and output data stores. For example, one job might extract data from two source systems into one EpiMart, and another job might extract data from a source system into external tables and from external tables into staging tables. In the second job, the external tables serve as both input and output data stores to different extractors.

THE DATA STORE DIALOG BOX

Each data store is represented by a Data Store dialog box (see Figure 56). This dialog box has tabs named General and Properties.

The **Data Store Type** pane of the General tab is where you enter the name and description for the data store and select its type: SQL Server, Oracle, Generic ODBC Data Source, or File.

*Note: The **Log** and **Datamart** options in the top-right corner are read-only and simply indicate when the opened data store is one of the built-in data stores.*

The **Data Flow** pane of the General tab has the following options:

- **Allow Use as Input Data Store** allows the data store associated with an extractor job step to be used as the input of an extractor.

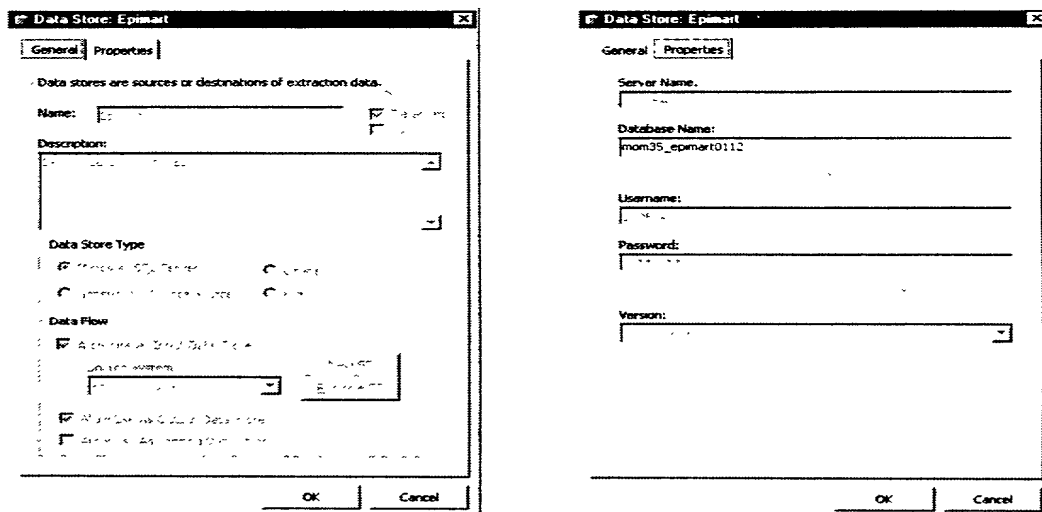
The source systems listed in the Source System drop-down list serve to distinguish source system identifier keys that may clash between different database systems. For example, if an account has an ERP and an SFA system, each of which has a Customer table, both may contain a customer number 100. To distinguish between these two records, two different source system identifiers must be used.

Note: *Fact rows join only with dimension rows that have the same source system identifier. For this reason, unless the site actually uses two or more logical source systems, select **Datamart Source** from the **Source System** drop-down list (the default).*

- **Allow Use as Output Data Store** allows the data store to be used as the output of a job step.

If this option is unchecked, an error message displays if the user tries to attach this data store.

FIGURE 56: DATA STORE DIALOG BOX: GENERAL AND PROPERTIES TABS



- **Allow Use as Logging Data Store**

This option ensures that logging is directed to the proper logging data store and must be selected in the Data Store dialog box.

E.piphany has special metadata tables for the purpose of logging its activity. Normally, this logging is written to the EpiMeta database. Logging to the EpiMeta, however, may increase its size significantly, so you may prefer to log to another data store. You can use the `logging_mssql.sql` script (included with the E.piphany software) to create a new logging database. If you run this script on another database, then a new data store can be used to log extractions.

The Properties tab has different fields, depending on the data store type. You can use the Properties tab to enter these fields:

- For SQL Server, enter the server's name, database name, username, and server's password and version.
- For Oracle, enter the Net8 name, username, the server's password, and the version number (Oracle8).
- For a generic ODBC data store, enter the DSN name and ODBC driver name, and server administrator's username and password. (You must also set up a DSN for the data store system using the Data Source Administrator in the Windows NT Control Panel.)
- For a data store of type file (`JobLogFile`), enter its file directory path.

MODIFYING THE DEFAULT DATA STORES

You need to modify the default data stores, which are `EpiMart`, `EpiMeta`, `JobFileLog`, and `LoggingDB`. Double-click their folders to open them and fill out the dialog boxes as follows:

- **EpiMart** specifies your EpiMart as a data store.

The only value that you can configure is the name of your EpiMart. You can use the Properties tab of the EpiMart Data Store dialog box to enter this database name, for example `Corp_EpiMart`.

- **EpiMeta** is the data store that references your current data, EpiMeta.
- **JobFileLog** specifies the data store for EpiChannel job logs.

In the General tab, the settings are correct for the default usage. The Data Source Type is a file, and the Data Flow is set to **Allow Use as Logging Data Store**.

Click the Properties tab, and change the directory for the JobFileLog from CHANGE ON INSTALL to the correct directory name. A valid directory name is required for a successful extraction. Leave the file name blank.

- **LoggingDB** specifies the data store for the EpiChannel logs.

In the General tab, the Data Source Type is your server, and Data Flow specifies: **Allow Use as Input Data Store** and **Allow Use as Logging Data Store**.

You can accept the default values for the log database's file name and directory path, unless you have located the log in a different database.

Note: *\$\$DEFAULT refers to the current EpiMeta.*

SETTING UP JOBS

The Job dialog box enables you to define jobs, create and order the job steps (extractors and system calls), assign data stores and roles, and schedule the job in a queue.

An EpiCenter has three default jobs—Incremental, Initial, and Merge—which are similar except for the semantics. The default jobs consist of the following:

- **Truncation:** a group with truncation steps for all of the built-in tables. You need to add truncation steps for the rest of the tables.
- **Default Extraction:** A group with empty groups for the fact and dimension table extraction.
- **job_type Semantics:** A group with empty groups for the dimension and fact semantics. There are groups for the Incremental, Initial, and Merge job types.

00625518.072500

- **Campaign Extraction:** A group with SQL-extraction steps and semantics for the backfeed tables (to move data from the backfeed tables into the datamart). You do not need to modify these SQL extraction steps unless one of the campaign-related tables has been changed.
- **AggBuilder**
- **MomBuilder**
- **e.4 End of extraction**, which consists of the following:
 - **MaxDate**—an empty group to which you need to add SQL to extract the date.
 - **Max of Maxdate**—SQL that finds the highest value extracted by MaxDate and records it as the last extract date.
 - **Campaign backfeed**—a group with special semantics that update the backfeed tables.
 - **Hotswap**—toggles the A/B/X/Y tables.
- **Refresh:** The system call to refresh the AppServer.

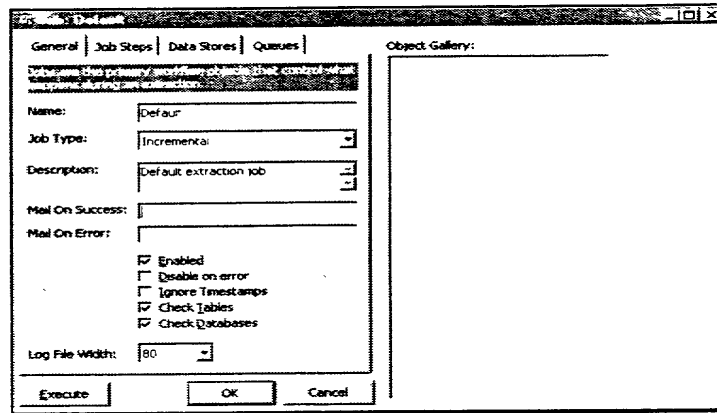
You may open a default job, rename it, and customize it, as well as create as many other jobs as necessary. To open a default Job dialog box, double-click it.

To open a new job dialog box, right-click the **Jobs** folder and select **New Job**.

The Job dialog box has four tabs: General, Job Steps, Data Stores, and Queues.

09625518.072500

FIGURE 57: JOB DIALOG BOX: GENERAL TAB



The General tab provides options you can set that control the following aspects of a job:

- For a job other than the default one, assign a name and description.
- Select a job type from the drop-down list. The options are described in “Job Types” on page 218.
- Assign addresses for e-mail notification of the job’s success or failure. See “Configuring E-Mail,” on page 243.
- Enable a job. EpiChannel executes enabled jobs only.

New jobs are initially enabled. You may, however, disable a completely functional job in some circumstances to accommodate system changes. For example, if a database is in the process of being moved or repaired, jobs that populate that database could be disabled as a protection against accidental execution.

- Request that the job be disabled if it encounters an error during execution.

- Request that all timestamps be ignored during job execution (**Ignore Timestamps**).

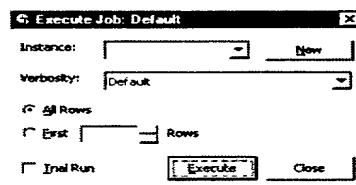
For example, by ignoring timestamps, you could retrieve all rows, without any date filters (based on EpiChannel's special filtering used for incremental extractions).

- Request that before executing the job, EpiChannel check that all databases referenced by any job databases and all EpiMart tables are available. Unless you are sure that this is true, select this option.
- Select the width of the log, either 80 or 132 columns. Select **80** for VGA monitors and **132** for SVGA monitors.
- Execute a single job.

Click the **Execute** button in the Job dialog box to execute this job. You can use the Execute Job dialog box (Figure 58) to do the following:

- Run the job as a trial run.
- Normally, the proper instance name is already selected. You can enter a new one by clicking the **New** button.
- Select a debug level that corresponds to the EpiChannel (**extract.exe**) verbosity levels (although you cannot specify row numbers). This debug level takes effect before the first SQL statement.
- Indicate the total number of rows to be transferred in a pull/push operation: all rows or the first N rows. This value is reset with each extraction statement.

FIGURE 58: EXECUTE JOB DIALOG BOX



Using the **Execute** button is equivalent to invoking the **extract.exe** command with this job as the job option. The **Execute** button, however, also supplies the database name, password, and so forth, if necessary (no instance is supplied) so that **extract.exe** does not depend upon the Registry entries.

JOB TYPES

Job types control how data is merged between the historical and current partitions of a fact table. Select one of these job types for each job:

- **Initial**

The Initial job type clears both historical and current partitions, and then allows you to load all newly extracted data into the historical partition. Select this type only when you are populating a newly initialized datamart, or replacing all of the data in an existing datamart. All data is moved into the historical partition, and the current partition is cleared. Lists are invalidated.

- **Incremental**

This job type allows you to add data to the current partition only. You cannot make changes to the historical partitions. You can add new or updated data to the current partition, although you cannot change any data that is already in the partition. Incremental jobs improve performance by reducing the copying of historical data. In addition, existing aggregates are updated only from the contents of current partitions. (See “Aggregate Building” on page 82 for more information about aggregates.)

- **Merge**

The Merge job type moves data from a current partition to a historical partition. New or updated dimension rows can be added, but no dimension rows can be changed. If historical fact data is changed, then aggregates are completely rebuilt. Otherwise, aggregates are updated without a complete rebuild.

09625518.072500

- **Rebuild**

The Rebuild job type allows you to make changes to fact and dimension tables in both current and historical partitions. All data is moved into the historical partition and the current partition is cleared. Aggregates are completely rebuilt.

- **Non-Partitioned**

Does not make use of the historical partitions. All data is kept in the current partition. Changes can be made to both fact and dimension tables. Do not assign this job type for a datamart that utilizes historical partitions, as all historical data is deleted. Aggregates are completely rebuilt.

JOB STEPS

The Job Steps tab is the main dialog box for job definition. In the right window of the dialog box you can define global objects for job steps and organize them into groups. In the left window, you can define the steps for a job. By selecting objects and dragging them onto job steps, you define the actual steps of the job.

The way you use this dialog box is described below:

1. Define a job step by clicking **New Step** in the Job Steps window of the dialog box. Enter the label name for a step and any description. (Job steps that you create locally appear in boldface, global extraction groups and nodes do not.) Click **OK**.

You can define a step but not have it execute by selecting it and clicking the **Disable** button, which toggles between **Disable** and **Enable**. A disabled step is identified by an X over its icon. To enable a disabled step, select it and click **Enable**.

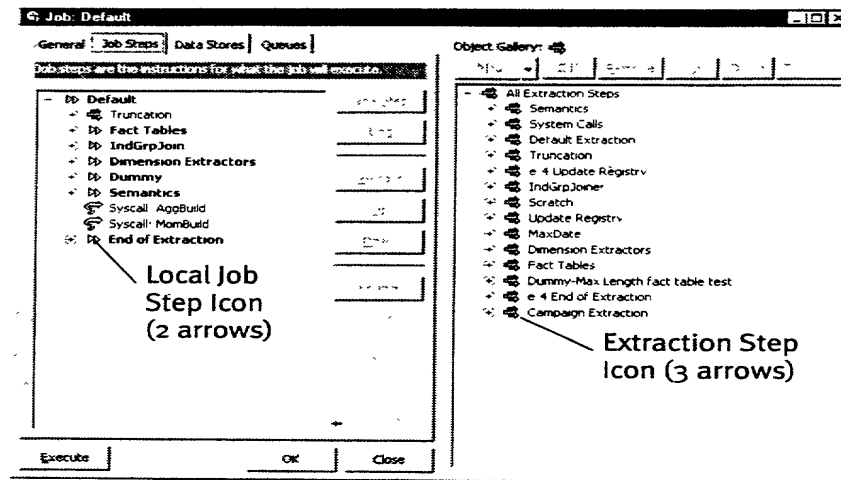
2. Order steps by selecting them and clicking the **Up** or **Down** buttons.
3. Clicking **Remove** deletes a selected step.
4. Click **Edit** to open a selected step in order to change its name and description.

- After defining objects in the **Object Gallery**, you can select and drag them onto the job step in the left window for inclusion.

The **Object Gallery** already contains the objects for the default job.

Note: A local job step icon consists of two arrows; an extraction step object icon consists of three arrows (see Figure 59).

FIGURE 59: JOBS DIALOG BOX: JOB STEPS TAB



To define a new global object:

- Use the **New** drop-down list in the right window of the dialog box to define groups into which you can place related job objects: SQL, semantic instance, system call, and truncation. All of these objects are at the same level.
- After you have created an object, you may select it in the **Object Gallery** and click **Edit** to modify its dialog boxes, or click **Remove** to delete it. These changes affect all occurrences of the object in the EpiCenter.
- You can also select an object and click **Up** or **Down** to reorder it in the list.
- Clicking **Duplicate** opens a dialog box for you to enter a new name for a copy of the object's definition. Double-click the new object name to open its dialog box for editing.

Note: You can disable individual parts of a global extraction group. These items are disabled for this job only.

DEFINING A NEW GROUP

To define a new group:

1. In the **Object Gallery**, either select a group (groups may contain groups), or select **All Extraction Groups** to place the group at the top level.
2. Hold down the arrow to the right of the **New** button and choose **Group** from the drop-down list.
3. In the dialog box, enter the group name and any description.
4. Indicate the action to be taken upon error (abort, ignore, or print).
5. After clicking **OK**, the new group is displayed as an object in the **Gallery**.

A group can contain other groups.

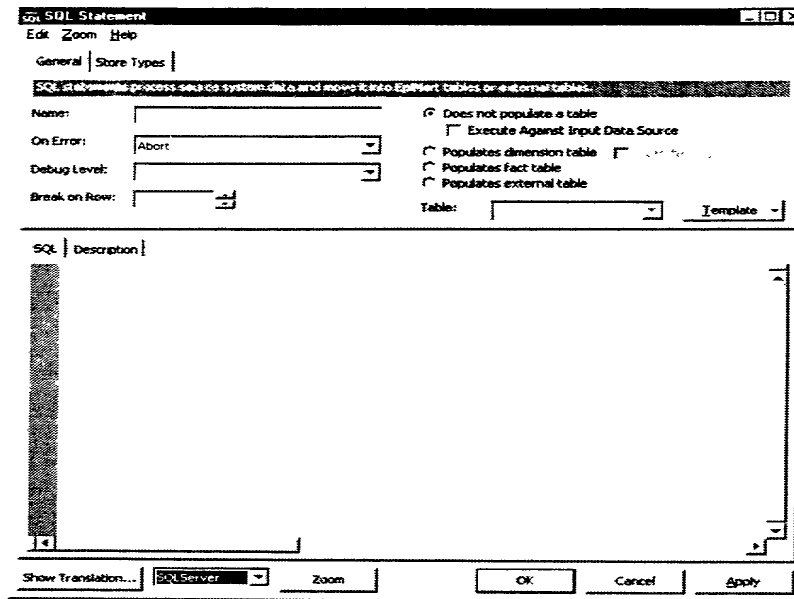
DEFINING SQL

If a job step requires SQL, EpiCenter Manager provides a template for the sample SQL. SQL statements process source system data and move it into EpiMart or external tables.

To define an SQL object:

1. Select the group for the new SQL in the **Object Gallery**.
2. Choose **SQL** from the **New** drop-down list. The **SQL Statement** dialog box (Figure 60) is displayed. It has two tabs: **General** and **Store Types**.

FIGURE 60: SQL STATEMENT DIALOG BOX: GENERAL TAB



In both tabs, the same SQL and Description tabs are in the lower pane. (You can use the description tab to add a description for your reference.)

3. In the upper pane of the General tab, enter a name for the SQL step, such as a step named **Customer Raw** for extracting raw customer data. When the step is added to the **Object Gallery**, **SQL:** is placed before this name to identify it as SQL.
4. Select the action to be taken if there is an error with the step. The default is to abort the step. (You can also ignore the error, or have it print to a log and continue.)
5. Select the debug level. These levels correspond to the verbosity levels on the **extract.exe** command line. See “EpiChannel Debugging Levels,” on page 116.

6. If you select a row number in Break on Row, execution stops at that row number for debugging purposes. If set to 0, the entire statement runs.

Note: You may set breakpoints on SQL statements that change the debug level before the SQL is issued. See “Setting Breakpoints,” on page 117.

7. Select whether the step:
- populates one of the following types of tables: a dimension table, a fact table, or an external table, or
 - does *not* populate a table (and if this is true, select whether it executes against input data store, or retains its default behavior of executing against the destination data store).

Most SQL statements are used to populate staging tables (or sometimes, external tables). An SQL statement may, however, be used to achieve a side-effect, in which case you can set it to “not populate a table.” In this case, the statement is executed, and any returned results discarded.

8. If the step references a table, select the table from the drop-down list.

Select this when SQL is to be expanded to reflect the structure of one of these tables, but the returned rows from the SQL statement are not meaningful. EpiChannel does not insert rows in the statement.

Extraction SQL is executed against the input data store and must be in the dialect of that database engine. The results are stored in the destination data store. EpiChannel automatically resolves SQL dialect problems if you use the EpiChannel database-vendor independent macros consistently in your SQL. (See Appendix A, “EpiChannel Macros” for more information.)

DISPLAY SAMPLE SQL

If the step populates a table, clicking the **Template** button displays sample SQL for the step in the lower pane of the SQL Statement dialog box. The SQL needs to be modified to contain the **FROM** and **WHERE** clauses appropriate for the tables in the source system, but the template lists the columns that must be returned. It does not matter in what order the columns are returned as long as they have the proper column names (and “extra” unused columns may be returned).

You can also view sample template code for a single column in a table. Holding down the small arrow to the right of the **Template** button opens a menu of the columns in the target table. Select the column whose template code you want to display. Select **All** to display code for all of the table’s columns.

You can have the system show the macro’s translation for your server. Click the **Show Translation** button and select your server type from the drop-down list.

The SQL Results window displays the SQL in color-coded text. The main menu commands enable you to use standard edit commands to search and replace lines, and you can set bookmarks on lines. The **Tools** menu has **Undo** and **Print** commands, and commands for clearing text with and without clearing the buffer.

You can resize the window, and click the **Zoom** button to have the text area fill up the entire SQL or Description window.

RESTRICT DATA STORES FOR THE EXTRACTION

You can restrict the input/output data stores for this extraction to Oracle and SQL Server. The statement is disabled for any unchecked type.

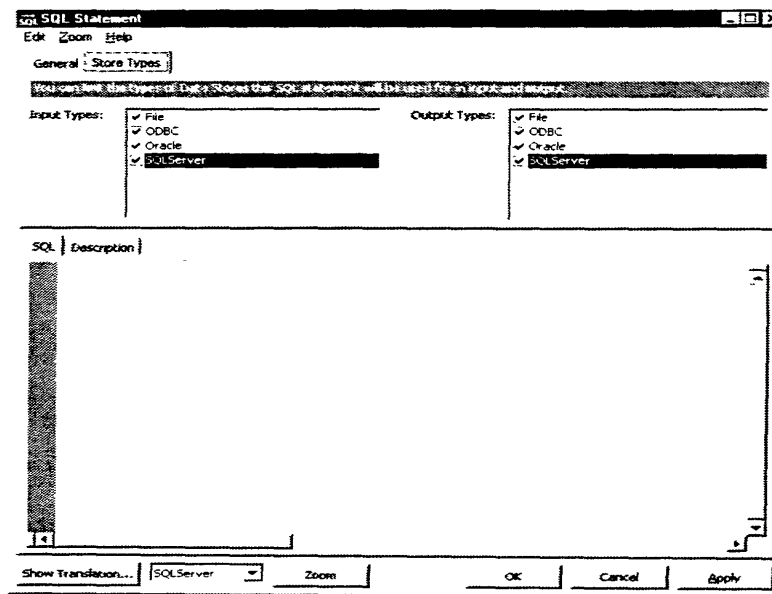
You may select multiple input types. For example, you could use the same SQL twice, once with SQL Server syntax and once with Oracle syntax, and have one or the other statements automatically disable itself when the source database is of the wrong type. This is of value when the group that contains the SQL statement is shared by multiple extractors.

Another approach to handling SQL dialect problems is to use E.piphany SQL replacement macros instead of database vendor-specific constructs. See Appendix A, “E.piphany Macros” for more information.

You can use the Store Types tab of the SQL Statement dialog box (Figure 61) to restrict data stores:

1. Check appropriate **Input Types**.
2. Check appropriate **Output Types**.

FIGURE 61: SQL STATEMENT DIALOG BOX: STORE TYPES TAB



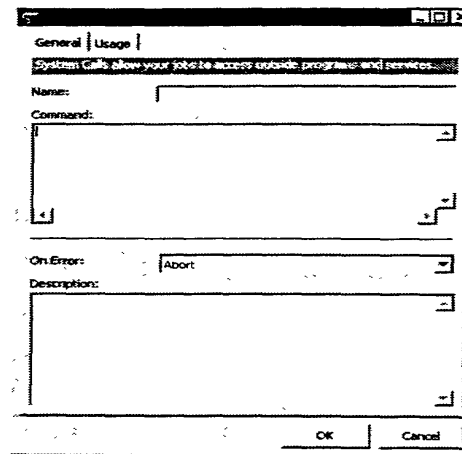
DEFINING SYSTEM CALLS

As mentioned, sites with more complex databases may require multi-stages and additional commands, such as lookup tables, aggregation splits, gathering data into ranges (binning), and duplicate detection. For this purpose, you may use system calls, which are executed during a job as if invoked from the console DOS command line.

To define a system call object:

1. Select the group folder for the system call object. The **System Call** folder is present by default although you can create system calls in any location.
2. Choose **System Call** from the **New** drop-down list.

FIGURE 62: SYSTEM CALL DIALOG BOX



3. Enter the name of the system call, the action to be taken upon error (abort, ignore, or print), the command line, and an optional description.

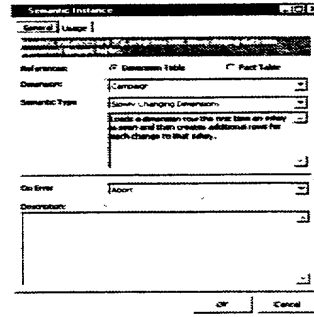
DEFINING SEMANTIC INSTANCES

To define a semantic instance object:

1. Select the group folder for the semantic instance.
2. Choose **Semantic** from the **New** button. (You can also right-click in the **Object Gallery** window and select **New Semantic** from the pop-up windows.) The Semantic Instance dialog box is displayed (Figure 63).
3. Select whether the semantic instance references a fact table or a base dimension table (the one to be operated on by the semantic type).

4. Select the associated table name.

FIGURE 63: SEMANTIC INSTANCE DIALOG BOX



5. Select the semantic type from the drop-down list. See Appendix F, "Semantic Types" for descriptions of semantic types.
6. Select the action to be taken upon error: abort, ignore, or print.

DEFINING A TRUNCATION OBJECT

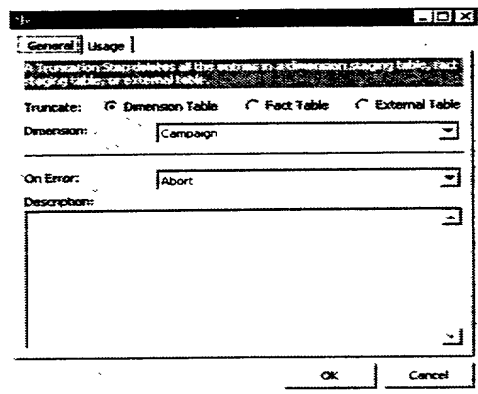
A truncation object can be added as a job step that truncates a specified table, either a dimension or fact staging table, or an external table. Typically, a job begins with the truncation (deletion) of old staging tables. Each table needs to be set up for truncation separately. You can create groups of tables to be truncated and reuse them in other jobs.

To define a truncation object:

1. Select the group folder for the truncation object.
2. Choose **Truncation** from the **New** menu in the Job Steps dialog box. The Truncate Steps dialog box is displayed (Figure 64).
3. Select the kind of table to be truncated (dimension, fact, external).
4. Select the name of the table to be truncated.

5. Select the action to be taken upon error: abort, ignore, or print.
6. Enter any description of this step.

FIGURE 64: TRUNCATE DIALOG BOX



DATA STORES TAB

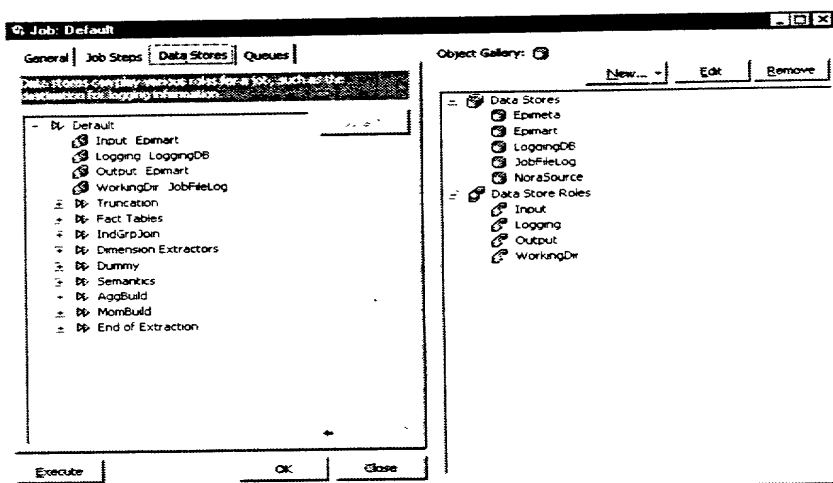
The Data Stores tab (Figure 65) of the Job dialog box allows you to assign the data stores and data store roles for a job and for job steps. The default data stores, data store roles, and any new data stores you have defined appear in the **Object Gallery**. For example, you can define custom data stores and data store roles and use them in system calls.

Assign a data store by selecting a data store object from the **Object Gallery** in the right window of the dialog box and dragging it onto its data store role in the left window. The object becomes attached. For example, select **Epimart** in the **Object Gallery** and drag it onto the **Output** role.

If a job step does not have a role assigned, it inherits the assignment from its first parent that does have a role assigned.

To remove an attached object, select it in the left window of the dialog box and click **Detach**.

FIGURE 65: JOB DIALOG BOX: DATA STORES TAB



THE SCHEDULER

The Scheduler determines when queues are run. *Queues* consist of tasks, which are jobs or events tied to saved reports, such as chart-file generation and campaign-list pulls. A *campaign-list pull* refers to the Scheduler's finding out which campaigns are due for export and then generating export files and backfeed tables based on these campaigns.

The Scheduler runs as a Windows NT service (as does the AppServer). After a standard installation, the Services control panel has a listing for *instance_name_scheduler*, in addition to *instance_name* for the AppServer. Normally, both of these services are always running.

To set up the Scheduler, follow these steps:

1. Choose **Configuration** from the **EpiCenter** menu. In the General tab of the Configuration dialog box, note these two **config_master** entries that govern Scheduler invocation.
 - **queue_wakeup_interval** specifies how often the Scheduler service reloads the queues to check whether any tasks are due to start. A *task* may be a job within a queue or a Chart or Campaign queue scheduled to be run.
 - **queue_start_precision** specifies how close the task's start time should be to the current time in order for the task to start.
 2. The default values are acceptable in most cases. If you need to modify them, select the key in the list and enter a new value in the **Value** text box. Increase these values only if you have many queues, and the time to process them (as evident from the log) approaches the value of **queue_wakeup_interval**.
- Note: The value of **queue_start_precision** must be greater than half the value of **queue_wakeup_interval**. Otherwise, a queue could fail to run completely.*
3. Click **Update** to change the configuration parameters.

SETTING UP A QUEUE

Double-clicking the **Queues** folder in the EpiCenter Manager tree opens the Queue dialog box (Figure 66). This dialog box enables you to define a queue and view its contents.

FIGURE 66: QUEUE DIALOG BOX: GENERAL TAB

General | Dependencies | File Types | Queue Contents | Logs | Usage

Queue Name: Campaign ☒ Enabled

☒ Refresh Metadata when this Queue starts

Queue Duration: ☒ As long as necessary
☐ For Days Hours Minutes

Start Time: 0:00

Occurring: ☐ Once
☒ Every hour(s) until 23:59

Per Task Duration: ☒ As long as necessary
☐ For Days Hours Minutes

Retry Overdue Tasks: ☒ Forever ☐ For Days

OK Close

To set up a Queue, open the General tab of the Queue dialog box:

1. Enter the unique name of the queue.
2. Check **Enabled** in order for the queue to be run. All queues are disabled by default.
3. Indicate whether or not you want the metadata to be refreshed when the queue starts.

Refresh metadata reloads all the metadata inside the Scheduler, similar to an AppServer refresh. The difference is that instead of the being associated with a Web page, a queue refresh is tied to a queue. Queues with Campaign and Chart file types need refreshing. For a job-only queue, in which metadata is loaded in a separate process, refresh metadata is an unnecessary overhead.

4. Indicate a duration of the queue: as long as necessary, or specify the number of days, hours, and minutes.

09625518, 072500

If **As long as necessary** is selected, the Scheduler does not terminate the queue. If a time is indicated, the queue runs this amount of time before the Scheduler service terminates it. This option is overruled, however, if a queue that can terminate the queue is scheduled to run.

5. Set the start time and indicate how often the queue should wake up: once, or every x number of hours until a specific time.

All default queues wake up once per day (8:00 p.m.). The maximum number of times that a queue can wake up is once each minute of the day, or 1440 times ($24 \times 60 = 1440$).

6. Indicate how long to retry tasks that are overdue: either forever, or for a set number of days.

If **Forever** is selected, whenever task execution is missed for any reason, the Scheduler attempts to execute the task the next time it wakes up the queue. When **For** is selected, the Scheduler re-executes the task only if its original execution date was no earlier than the specified number of days.

Note: Reschedule your queues for Daylight Savings times so that they do not start during invalid HHMMSS combinations.

You can use the Dependencies tab (Figure 67) to set up dependencies between this queue and other queues expressed in parent/child relationships. Parent queues must run before child queues. Child queues can run only after the parent queue finishes. If a child queue is already running, however, and the parent queue becomes ready to run, then the parent has to wait for the child to finish (unless the parent queue is set to terminate the child queue).

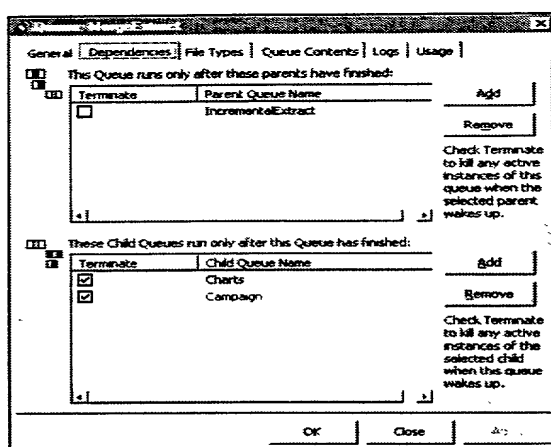
Note: When making timing and dependency selections, estimate how long your queues might run and the optimal time for them to run. Also, consider whether queues might interfere with each other.

In the upper pane, click **Add** and select any child queues that depend on the queue you are defining as a parent queue. If you check **Terminate** next to a child queue, any instance of this child queue that is running when the parent queue is ready to start is terminated. A parent queue cannot start if a child queue is running. It has to either wait for the child queue to finish, or terminate it.

In the lower pane, click **Add** and select any parent queues of which this queue is a child. Check **Terminate** to kill any instances of this queue that are running when the parent starts running.

Note: *Before modifying any dependencies, stop the Scheduler. After setting up dependencies between queues, restart the Scheduler.*

FIGURE 67: QUEUE DIALOG BOX: DEPENDENCIES TAB



To view the contents of this queue, click the Queue Contents tab (Figure 68). A task in the queue is shown by name, next schedule type (such as **As Soon As Possible**), next date (if appropriate), and the number of times that the task is expected to run during the day.

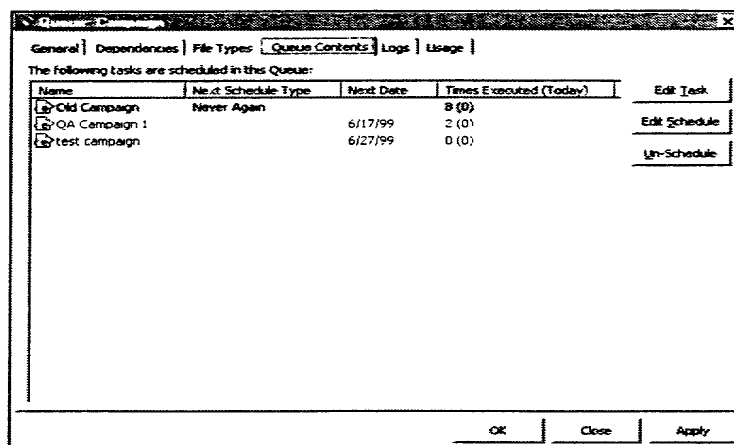
Note: *As Soon As Possible appears for newly added tasks. The first time that a queue starts after a task has been added, As Soon As Possible is replaced by a value in the Next Date column.*

You can use this dialog box to remove a task from the queue by selecting it and clicking **Un-Schedule**.

To edit the queue dialog box for the job, click **Edit Task**.

To edit the schedule for this queue, click **Edit Schedule**, which opens the Task Schedule dialog box (Figure 70, on page 237).

FIGURE 68: QUEUE DIALOG BOX: QUEUE CONTENTS TAB

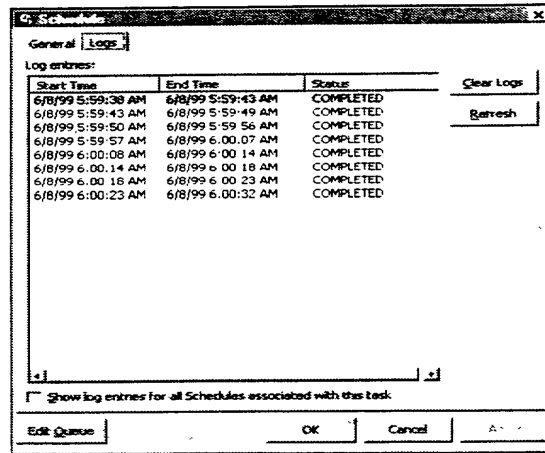


The Logs tab of the Queue dialog box shows logs for the queue. A log entry contains the start and end times of the queue and its status: failed, completed, server aborted, and so forth. These logs report on whether the queue ran successfully, not whether members of the queue ran successfully. Therefore, a queue log can report that the queue ran successfully, although a task that ran as part of the queue failed.

To view the task logs, open the Queue Contents tab, select the instance of the task, and click **Edit Schedule**. In the Task Schedule dialog box (Figure 70, on page 237), click the Log tab to see the task logs.

The logs shown are for all instances of a task that have been run by this queue or that are about to be run by this queue (that is, all instances that are supposed to run as part of a currently running queue). Some scheduled tasks, such as jobs, can be in multiple queues. When **Show entries for all Schedules associated with this task** is unchecked, only instances associated with this queue are displayed. When the check box is checked, all instances, including ones associated with other queues, are displayed.

FIGURE 69: TASK SCHEDULE DIALOG BOX: LOGS TAB



Clearing the logs for a queue does *not* reset the internal execution history. If a daily task has already run, and you clear the logs and re-set the daily run-time for later in the day, the task does not run again at the scheduled time, because it has already run for today.

SCHEDULING JOBS WITHIN QUEUES

Note: Use the **Queues** folder to define queues. For instructions, see “Setting Up a Queue,” on page 231.

The Queues tab of the Job dialog box allows you to assign a job to one of the queues you defined in the **Queues** folder and to schedule this task in relation to other task in the queues. When you drag a job queue into the left window of this dialog box, the Task Schedule dialog box (Figure 70, on page 237) is displayed.)

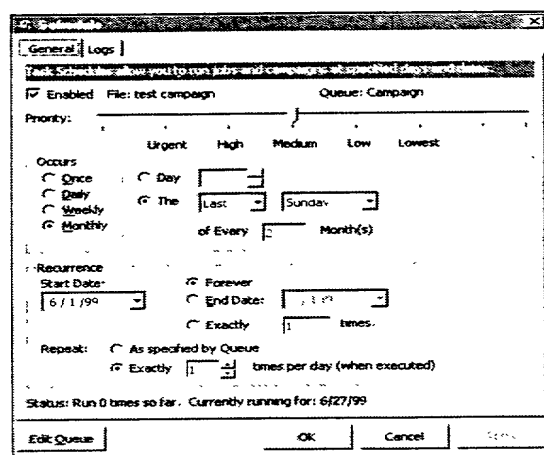
THE TASK SCHEDULE DIALOG BOX

The Task Schedule dialog box (Figure 70, on page 237) allows you to schedule tasks within queues (a *task* is any job, Campaign, or Chart). This dialog box is displays when you assign a job to a queue in the Job dialog box, and when you edit the schedule for any task. You can use the Task Schedule dialog box (Figure 70) to assign a priority and run-time parameters to tasks within a queue. The actual time that the Scheduler runs the queue runs depends on its relationship to other queues.

Note: Preliminary values for each Campaign or Chart are assigned based on what the end user specified, but you can modify these.

You can open a Queue dialog box for editing by clicking the **Edit Queue** button in the Task Schedule dialog box.

FIGURE 70: TASK SCHEDULE DIALOG BOX



Follow these steps to schedule a task within a queue:

1. Assign this task a priority in relation to other tasks in the same queue. When you move the slider, the integer priority value, which ranges from 0 to 100, is displayed.
2. If **Enabled** is checked, the task can be executed.
3. Indicate how often the task occurs (once, daily, weekly, monthly).
4. Set starting and ending dates, or specify the number of times that the task should run. When you specify an exact number of times for it to run, the current number of times actually run is shown in this dialog box (after it has run at least once with this schedule).

THE CAMPAIGN AND CHART QUEUES

File types reflect the association of a report with a queue. The campaign file types in the default Campaign Queue are:

- GroupCampaignFile
- GroupCampaignListFile
- IndividualCampaignFile
- IndividualCampaignListFile

Whenever a campaign is saved, the system places it into one of the system-defined file types associated with a Campaign queue. Similarly, the system assigns Chart file types to Chart queues. The default Chart queue has the file type, TicksheetFile.

Note: Do not change the default file types in the File Types tab of Campaign- and Chart Queues dialog boxes. They are pre-configured.

When the queue wakes up, it finds all saved reports of this type that were scheduled to run at least once more today, or that have not run as often as they should have on a previous day (but have not yet expired). All of the overdue, but unexpired, task instances are put into the queue, along with one instance of every task that needs to run today. So, for example, if a task is supposed to run four times a day, but it ran only once yesterday, and not at all today, then assuming that tasks expire after more than a day, when the queue wakes up, it plans to execute all three of yesterdays' missed task execution, plus one more for today.

Running a Campaign queue results in campaign export. Seeds are included in campaign output only after the Scheduler runs a Campaign queue.

EXTERNAL TABLES

Extraction statements are sometimes directed to load external tables that serve as intermediary tables for multi-staged extraction. Epiphany supplies one external table called `last_extract_date`. Use of this table is recommended, but optional. See “External Tables,” on page 76.

To define external tables and their columns:

1. Right-click the **External Tables** folder and select **New External Table**.
2. Enter the name and any description for the table in the External Table dialog box (see Figure 71).
3. Select the Data Store for this table from the drop-down list.

An external table must be in the Epimart datamart store in order to be generated.

Click **New** to create a new data store as described in “The Data Store Dialog Box,” on page 211.

4. By default, the **Generate** option is checked.

Check **Generate** if you want to make the external table the target of an extraction statement and prefer to have the schema generator generate the table. This saves you from having to manually enter the column names. Unchecking this option means that you have to add the columns (as described below) to create this table.

09625518.072500

FIGURE 71: EXTERNAL TABLE DIALOG BOX

External Table: Indigipacts

General

An external table is a temporary table that is normally used during a data load or export operation.

External Table: Indigipacts

Data Store: Epmart New...

Description:

☒ Generate table (If unchecked then columns are not specified below)

Column Name	Physical Type	Null...	
amount	MONEYSTRING	N	Add Column Edit Remove
date_key	SMALLDATE	N	
ID	VARCHAR_25	N	
name	VARCHAR_50	N	
product	VARCHAR_50	N	
transtype	VARCHAR_15	N	
units	FACTQTY	N	

OK Cancel

To add a column:

1. Click **Add Column** and enter the name in the External Column dialog box (Figure 72).

FIGURE 72: EXTERNAL COLUMN DIALOG BOX

External Table: Indigipacts: External Column

General

An external column is a physical column in an external table.

Name:

Physical Type: VARCHAR_100 Nullable ☒

Description:

OK Cancel

2. Select the physical type. See Appendix D, “Physical Type Values” for descriptions of these physical types.
3. If **Nullable** is checked, then the external column allows null values.
4. Add as many columns as necessary. After you click **OK** in this dialog box, the columns are added to the **Column Name** list in the **External Columns** pane of the External Table dialog box.

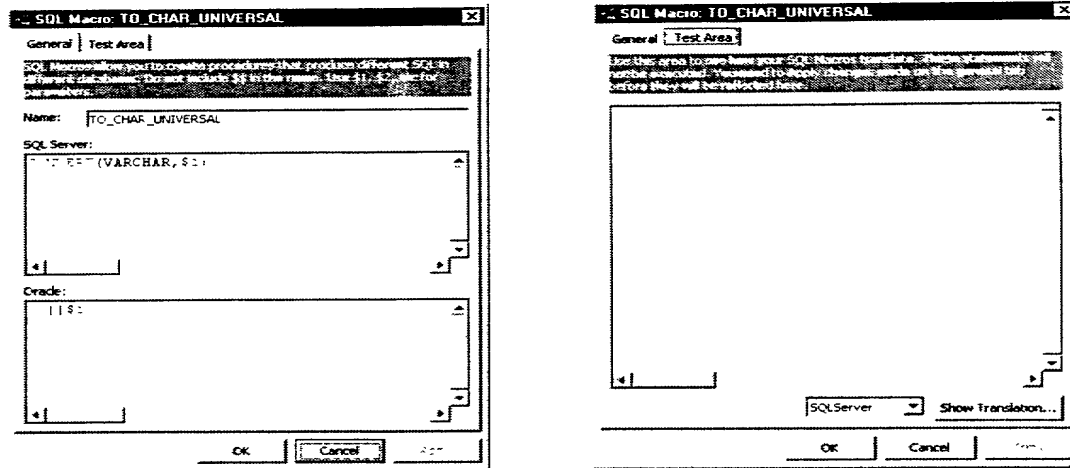
MACROS

E.piphany provides optional SQL macros as described in Appendix A, “E.piphany Macros.” You may also use EpiCenter Manager to create your own SQL macros. Follow these steps:

1. Right-click the **Macros** folders and select **New Macro**.
2. In the General tab of the SQL Macro dialog box, enter the macro name without the symbol \$\$.
3. Enter the SQL syntax for both SQL Server and Oracle, unless you want the macro to translate to nothing in one database.
Reasons for not translating a macro are that it not be necessary for one vendor (the \$\$NO_FROM_LIST macro on SQL Server), or it may be specific to a vendor (the \$\$ORACLE macro).
4. Click the Test Area tab to view the macro’s translation.
5. Select whether you want to display the translation for SQL Server or Oracle, and click **Show Translation**.

When viewing the translated SQL, note that any occurrences of the \$\$ symbol are highlighted in red to make them easy to find. These macros did not translate because they do not exist, often because of a typo or other error.

FIGURE 73: SQL MACRO DIALOG BOX



LIST MANAGER EXTRACTION

To run List Manager, configure an appropriate default job in the **Jobs** folder. When List Manager is installed, the order of the extraction should be as follows:

1. SQL/Semantics
2. AggBuilder

Note: Aggbuilder **must** run before MomentumBuilder in merge and rebuild jobs.

3. MomentumBuilder
4. e4 End of Extraction

5. After finishing the extraction, refresh the AppServer as described in Chapter 5, "The E.piphany Application Server."

For information about running MomentumBuilder as a stand-alone executable and to determine if MomentumBuilder extracted the correct data, see "The MomentumBuilder Program," on page 111.

CONFIGURING E-MAIL

To have the system automatically send e-mail notification of the outcome of a job (either successful completion or job failure), EpiCenter Manager needs to know your e-mail Profile name and password. These must match the information in the **Mail and Fax** or **Mail Control Panels**. Windows uses this Control Panel to define the profile of mail accounts, where each profile could tie users to different mail servers. The operating system then makes mail to this server possible via the **SMAPI** program interface used by EpiChannel.

To determine your e-mail profile name, follow these steps:

1. Open the **Mail and Fax Control Panel** (from the **Start** menu, choose **Settings\Control Panel** and double-click **Mail and Fax**).

You may be given a list of profiles or may be placed into the only profile available. If you do not remember your profile name, you can click **Show Profiles** to display the names of the existing profiles.

2. Choose **Configuration** from the **EpiCenter** menu.

The list of EpiCenter configuration variables includes **Mail Password** and **Mail Profile Name**.

3. Click **Mail Profile Name** and enter the name of the profile as shown in the **Mail and Fax Control Panel** in the **Value** text box. Click **Update**.
4. Click **Mail Password** and enter the password. Click **Update**.

The mail password depends upon which mail server is used and follows the rules of this server. Often, the password is the same as the Windows NT user password.

Note: Because this password may be visible to others in mail logs or in metadata exports, do not use a password of any importance. Create a new mail user on your mail server exclusively for EpiChannel if revealing this password presents a problem.

VERIFYING THAT E-MAIL NOTIFICATION WORKS

To test if e-mail is configured correctly, follow these steps:

1. Open a Job dialog box for a job that simply runs and exits. In the General tab, enter the addresses for mail on success and mail on failure.
2. Run the job.

If you receive e-mail notification, you have set up EpiChannel e-mail correctly. If not, repeat the steps for configuring e-mail given above and run another job. Please contact EpiChannel Customer Support if there is still a problem.

CONFIGURING OUTLOOK EXCHANGE FOR EPICHANNEL

Follow these steps to configure Microsoft Outlook Exchange 98 for EpiChannel e-mail notification:

1. Install Microsoft Outlook Exchange 98 according to the instructions on the screen. When prompted to select which kind of installation, select **Corporate\Workgroup**. Reboot your computer.
2. Start Outlook Exchange and choose to configure an Internet Mail account.
3. When prompted, enter a **Profile Name** for this account.

Note: The Mail Profile in the EpiCenter Manager Configuration dialog box is the default e-mail address for the EpiCenter Manager user.

4. Set up the account with an appropriate e-mail account name, user name, and organization.
5. For the e-mail and reply address, enter a bogus address, such as *bogus@xyz.com*.

6. In the Servers tab, set both the incoming and outgoing mail servers to the name of the mail server that can route mail outside the company's firewall. (This name is not case-sensitive.) In the Incoming Server section, enter any account name and password. (Because EpiChannel never uses Outlook Exchange to check incoming mail, these can be any values.) Accept the default values for the other tabs.
7. Exit Outlook Exchange.

Note: If the mail server is installed on a UNIX platform, you can test it by logging on with a shell account and sending mail to yourself and Epiphany from the UNIX mail program.

PURGING EPIMART TABLES

You can remove tables from the EpiMart that are no longer needed. For example, you may have unused tables to delete or, as a result of metadata changes, have decreased the number of aggregates. These higher numbered aggregate tables remain in effect until you purge the EpiMart tables.

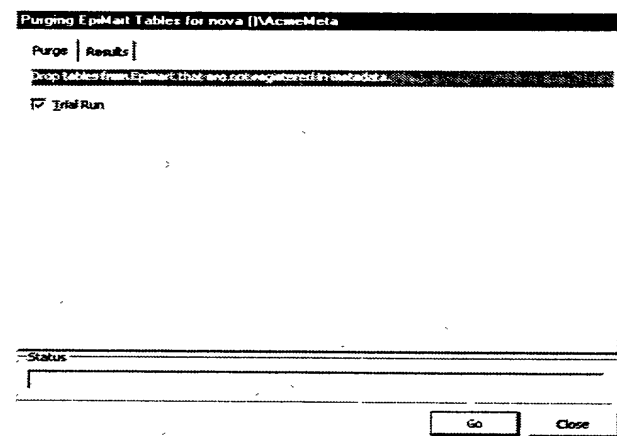
You should first try a trial run.

To purge database tables:

1. Choose **Purge Epimart Tables** from the **EpiCenter** menu.
2. Check **Trial Run** in the **Purge** tab, and click **Go**.

After the run, click the Results tab to view the tables to be deleted. If these results are acceptable, return to the Purge tab, uncheck Trial Run, and click Go to delete these tables from your EpiMart.

FIGURE 74: PURGE EPIMART TABLES DIALOG BOX



MEASURES

A *measure* is a business calculation that is the result of an arithmetic combination of fact columns. Each EpiCenter has associated measures, which are organized alphabetically by name in the EpiCenter Manager's **Measures** folder. At the frontend, a user selects measures on a Web page. The calculations that apply to this query depend on how you define the measures.

The **Measures** folder consists of the sub-folders: **Measures**, **Measure Layouts**, **Measure Sets**, and **Transaction Filter** sets. To define measures for the EpiCenter, follow these steps:

1. Right-click the **Measures** folder and select **New Measure** from the pop-up menu. The Measure dialog box (Figure 75) is displayed.

FIGURE 75: MEASURE DIALOG BOX: GENERAL TAB

New Measure: ASPAdiGrossMonth

General | Measure Terms | Usage

Measures are business calculations made from arithmetic combinations of fact columns.

Measure Name: ASPAdiGrossMonth Units: UNITS

Measure Label: ASPAdiGrossMonth

Description:

OK Cancel Apply

2. In the General tab, enter a name for the new measure.
3. If the label (the name visible to end users) differs from the name, replace the label name.
4. Enter a description of this measure (for your reference only).

5. Select the unit of measurement for the calculations from the **Units** drop-down list. A default EpiCenter configuration has these measure units:
 - **Currency_Local** (for local monetary units, such the Franc or Yen).
 - **Currency_US** (for currency expressed in U.S. dollars)
 - **Percent** (for percentages)
 - **Units** (for the count of an item)

You can define unit options via the Measure Units tab of the EpiCenter Manager's Configuration dialog box (see "Measure Units," on page 420).

MEASURE TERMS

You can use the Measure Terms tab of the Measure dialog box (see Figure 76) to define how the measure is calculated. A *measure term* is one component of an arithmetic expression that makes up a measure. A measure term refers to the aggregation of a single fact column in a fact table, such as `SUM(Order.net_price)` with a particular transaction type (or transaction type set). Each term applies to a specific column in a fact table, transaction type or set, and backlog type (if applicable).

09625518.072500

FIGURE 76: MEASURE DIALOG BOX: MEASURE TERMS TAB

Measure: ASPAdjGrossMonth

General | Measure Terms | Usage

Operator / Constant:
-SUM

Fact Table / Column:
SellThruMonth
net_price

TransType / Trans Set:
SHIP

Backlog Type:

Dimension Role / Column:

+ - * /

Add Remove Down

#	Op / Const	Fact Table	TransType / Set	Backlog	Fact Column	Dimension Role	Dimension Column
1	-SUM	SellThruMonth	SHIP		net_price		
2	-SUM	SellThruMonth	SHIP_ADJUST		net_price		
3	add						
4	-SUM	SellThruMonth	SHIP_ADJUST		number_units		
5	-SUM	SellThruMonth	SHIP		number_units		
6	add						
7	div						

Preview:
SellThruMonth net_price + SellThruMonth net_price) / (SellThruMonth number_units +
SellThruMonth number_units)

Show Preview with:
☐ Operators
☐ TransTypes
☐ Backlog Types

OK Cancel Apply

One measure term is combined with other measure terms to create a composite measure. Each defined measure term is a numbered step that displays in the lower pane of the Measures Terms tab. The E.piphany system converts these steps to appropriate SQL SELECT statements.

Note: Reverse Polish Notation is the mathematical notation used to construct measure definitions. See “Reverse Polish Notation,” on page 252 for more information.

Use the upper pane of the Measure Terms tab to define a step for a measure term. After defining the term, click **Add** to add it as the next step in the lower pane:

- 1. Select an operator.

Either select one of the SQL operators from the drop-down list: **SUM**, **MIN** (minimum values on a fact column), **MAX** (maximum values on a fact column), **COUNT**, **COUNT DISTINCT** (or the negative values of these operators, such as **-SUM** or **-COUNT DISTINCT**), or enter a constant character value in the text box.

When counting operations are performed on dimension tables, the effect of slowly changing dimensions can result in duplicate counts. To avoid this, use a measure that performs a `COUNT DISTINCT` on the fact table with reference to the unique ID column of the dimension role (assigned when you created the demographic base dimension table). Otherwise, in a slowly changing dimension, the `COUNT DISTINCT` operator would count a single element each time that it appears in the dimension table.

2. Select the fact table and column where the data resides.
3. Select an associated transaction type, or one of the transaction type sets defined as part of your schema.

All facts have an associated transaction type. For a description of the kinds of transaction types, see Appendix B, “EpiCenter Configuration.”

If you select a Transaction Type Set, a single measure term calculation (usually `SUM`) can be used to add up all rows for all transaction types in the set.

4. Select the backlog type, if appropriate.

The backlog types are **BEGIN** or **END**; or leave blank if the backlog type does not apply. You can use a backlog type when a measure term (a single line of the measure definition) should exhibit accumulating behavior. Normally, when running a report of Sales by Month, for example, the report shows only the sum of transactions that occurs in each month of the report.

When a backlog type is used, however, the columns of the report show accumulated values from previous months, in addition to the current month. You can use **BEGIN** to show the accumulated value at the beginning of each period, and **END** to show the ending accumulated value.

For example, assume that report of sales by month transactions shows \$10 for June, \$20 for July, and \$40 for August. A report of the beginning backlog shows the accumulated value at the beginning of each month:

June	July	August	September
\$0	\$10	\$30	\$70

A report of the ending backlog shows the accumulated value at the end of the month:

June	July	August
\$10	\$30	\$70

Note: *Backlogs are always calculated based on the built-in date dimension role. User-defined dimension roles that refer to the date dimension are not available for backlog calculations.*

5. For COUNT DISTINCT operators only, select the associated dimension role and column (which should have a unique ID column).
This enables the counting of the distinct values for a specific column in a dimension table related to Campaign Manager or List Manager.
6. Click **Add** to add this as the first step in the **Measure Terms** pane.
7. If appropriate, add another measure term by repeating the above steps.
8. Click the appropriate arithmetic operator to be applied to the measure terms: add (+), subtract (-), multiply (x), or divide (/). See “Reverse Polish Notation,” on page 252 for examples.

The **Preview** pane of the Measure dialog box (see Figure 80) shows the translation of the Reverse Polish Notation calculations to standard (infix) calculations. If the measure calculation does not currently make sense (for example, a term is followed by two adds), the notation in the lower pane displays in red with one or more missing or extra term tags. You can save the measure, but the AppServer cannot use it.

You can choose to display only operators, transtypes, or backlog types, or any combination of these in the Preview.

Warning: *The removal of fact or dimension columns from the EpiCenter can remove measure terms, which invalidates their calculations. In this case, check all of the measures to determine if there are any Preview displays that appear as red.*

To edit an existing measure, double-click its **Measure** folder, which displays the Measure dialog box. Modify this as described above. Click **OK** to save your changes.

To delete a measure, right-click its folder and select **Delete** from the pop-up menu. You can also use this pop-up menu to duplicate or export the measure.

CONFIGURING MEASURES FOR SCORING

You can use measures directly to score a list using the Scoring Web page. When configuring your EpiCenter, consider the kinds of measures you need to define for use with scoring. Some commonly used measures for RFM (Recency, Frequency, Monetary) analysis are as follows:

- The length of time since a customer purchased an item.
- The number of times a customer purchased the item.
- The total amount of money a customer has spent.

REVERSE POLISH NOTATION

The **Measure Terms** pane in the Measure dialog box uses Reverse Polish Notation (RPN) to construct measure definitions. (Reverse Polish Notation is named for its Polish inventor, Jan Lukasiewicz.) RPN operations are performed in a last-in, first-out (LIFO) basis. All of the values to be operated upon are placed in a stack. Then the top two are operated upon and the result of that operation is placed in the stack, replacing the previous two values. Then the next top two are operated on and the result placed in the stack, and so forth.

For example, using Reverse Polish Notation for this calculation:

$$1 + (2 * 3) = 1, 2, 3, *, +$$

means that 1, 2, and 3 are placed in the stack. The last two items (3 and 2) are multiplied, and the resulting value 6 is placed in the stack. The stack now holds 6 and 1, which are added, and the result 7 is placed in the stack.

In contrast, applying RPN to the calculation:

```
(1 + 2) * 3 = 1, 2, +, 3, * in RPN
```

means that 1 and 2 are placed in the stack. These items are added, and the result 3 replaces them. Next, the value 3 is placed in the stack (the stack now holds 3 and 3). These items are multiplied and the result is 9.

The following example shows how RPN is used to define a measure definition (in the Measure dialog box). The Average Sales Price (ASP) for Booked/Gross Orders equals the total number of dollars received, divided by the total number of units shipped.

```
SUM (Order.net_price)
SUM (Order.number_units)
div
```

This example is calculated using RPN as follows. The sum of all of the Order net prices is calculated and placed in the stack. Then the sum of all of the number of units is calculated and placed in the stack. Next, the division operator is applied to the top two items in the stack. The result equals the ASP.

For the operators that apply to the aggregates, use the arithmetic operators: add, sub, mult, and div.

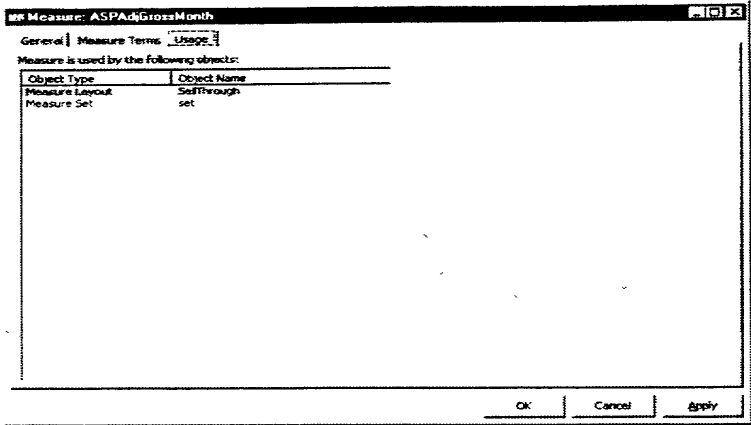
The next example a similar calculation. This time the ASP is calculated for booked net orders (booked orders minus returns). Here the sums of two Order net prices are added (the returned items are represented as a negative number) and placed in the stack. Then the sums of two Order number of units (the returned items are represented as a negative number) are added and placed in the stack.

```
SUM (Order.net_price) BOOK
SUM (Order.net_price) BOOK_RETURN
add
SUM (Order.number_units) BOOK
SUM (Order.number_units) BOOK_RETURN
add
div
```

DEFINING MEASURE USAGE

To view the objects that use the measure, click the Measure Usage tab of the Measure dialog box (Figure 77). This dialog box tells you what other objects use this Measure object. (Editing or removing a global object affects all usages of the object.)

FIGURE 77: MEASURE DIALOG BOX: USAGE TAB



MEASURE LAYOUT

Defining the measure layout for a Web page is a two-step process. First, you can add all of the measure column elements that you want to appear on the final Web page and arrange them by column. This sets up the organization of the Web page and assigns names to the measure column elements in the Web page (the actual value is derived from the measure to which you map this combination of elements).

Second, you need to map each set of measure column elements that a user might choose from the Web page to the measure calculation that this choice represents. (Measure layouts apply to all reporting and analysis Web page types except Influences and Cluster, which use measure sets instead.)

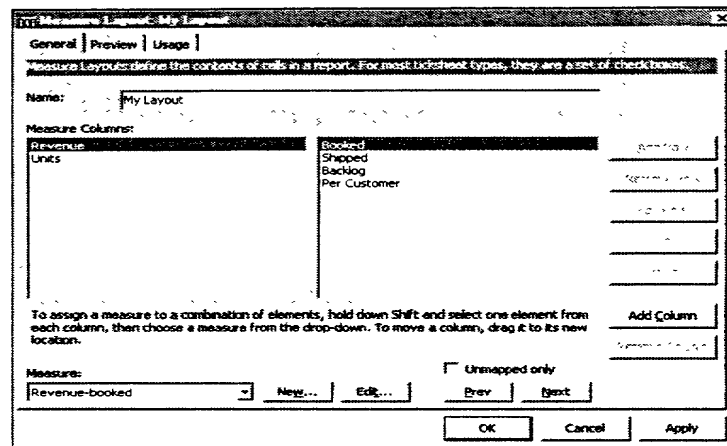
You can create a new measure within the Measure Layout dialog box. Click **New Measure**, which opens the Measure dialog box (see Figure 75). Follow the instructions given in “Measures,” on page 246.

ADDING ELEMENTS TO COLUMNS

To add elements in the appropriate columns:

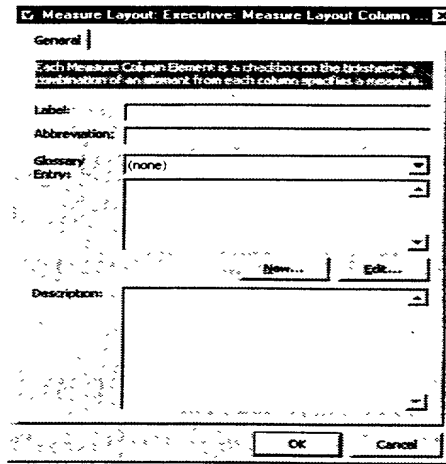
1. Open the General tab (Figure 78) in the Measure Layout dialog box. Note that a single empty column is displayed for a new measure layout.
2. Click **Add Row** to define the measure column elements for this column.

FIGURE 78: MEASURE LAYOUT DIALOG BOX: GENERAL TAB



3. In the Measure Layout Column Element dialog box (Figure 79), enter the label (the name as it appears in the Web-page column), an abbreviation (if appropriate), and a description for your reference.

FIGURE 79: MEASURE LAYOUT COLUMN ELEMENT DIALOG BOX



4. To set up a glossary entry, click **New** and type the entry name and help text (that displays to end users) in the Glossary dialog box (see Figure 82).

Note: The description you enter in the Help Text box can include HTML coding for links. For an example, see “Glossary Entries,” on page 262.

By defining glossary entries, measure column elements in Web pages, such as Units, Gross, and Sell-Through, are hyperlinked to a glossary page. When a user clicks a link, a glossary page displays which defines it. These glossary entries help users to understand your terminology.

5. Click **OK** to add the measure column element to this column.
6. Click the **Add** button again to add another measure column element to the column. Continue to add elements as described above until you have entered all of the elements for the column.
7. Click the **Add Column** button to add a second empty column. Click **Add** to enter the new column’s measure column elements as described above.

8. Click **Add Column** to add another empty column, if applicable. There is a maximum of five columns per Measure Layout. Add the measure column elements to the new column.
9. To change the order of one column with another (the order in which they display on the Web page), select a column and drag it onto the new column location. The columns' contents are switched.
10. To remove a column, select it and click the **Remove Column** button.

MAPPING ELEMENTS TO A MEASURE

End users can select any combination of measure column elements (one per column) on a Web page. Each combination of elements equals a measure (whose calculations determine the contents of the generated report or query). For each Web page, you need to map every measure column element in a given column to every other measure column element in all of the other columns (to cover all of the possible combinations that a user may select).

To map elements to a measure:

1. While holding down the **Shift** key, select an element from each column. Each selection is highlighted.
2. With all of your selections highlighted, choose a measure from the drop-down list.

This measure is invoked by the system whenever the user selects this combination of elements on the Web page.

VERIFYING THAT ALL ELEMENTS ARE MAPPED

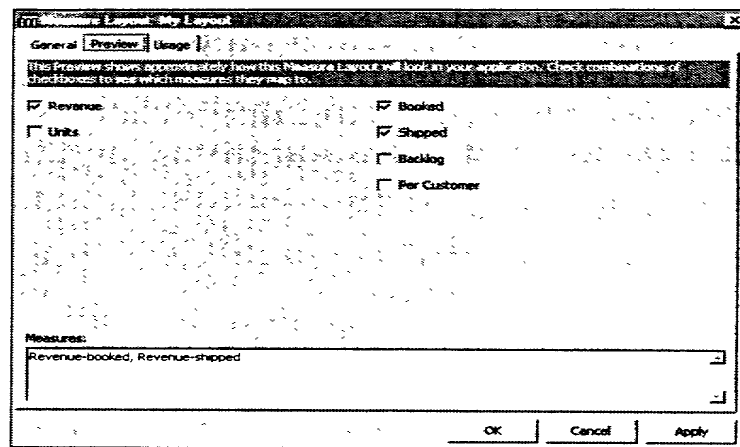
Click the **Previous** and **Next** buttons to cycle through the combinations of measure column elements that comprise a measure. The mapped element in each column is highlighted, and the measure that these elements map to is shown in the Measure text box.

Check **Unmapped Only** to display only those combination of elements that have *not* been mapped. (Remember that each distinct path through the columns must be mapped.)

Click the **Previous** and **Next** buttons to cycle through the combination of elements. Unmapped elements are highlighted, and no measure name is displayed in the Measure text box.

You can use the Measure Layout Preview tab to view how the element layout appears on the Web page. Make one or more selections per column; the associated measure or measures are shown in the Measures text area.

FIGURE 80: MEASURE LAYOUT DIALOG BOX: PREVIEW TAB



MEASURE SETS

Instead of measures, Influences and Community Clusters Web pages use measure sets (one or more measures that work in combination as a set). There are three types of measure sets: Classification, Regression, and Clustering. Influences Web pages use the Classification and Regression types. Community Clusters Web pages use the Clustering types.

To answer end-user queries, the E.piphany system builds models that use two types of trees: classification and regression. A *classification tree* finds rules that can predict the value of a discretely valued *attribute* based on the values of a set of other attributes. A *regression tree* finds rules that can predict the value of a particular numeric *measure*, such as customer profitability, based on the values of a set of attributes, such as customer attributes. The difference between classification and regression is that classification trees use attributes to predict an attribute, and regression trees use attributes to predict a measure. The basic questions answered by the two types of trees are very similar, but they require slightly different configuration.

Clustering is the process of finding groupings in data. The E.piphany system identifies groups as places of high concentration of data points. These groups are usually defined by some kind of internal consistency, such as households with similar demographics, or individuals with similar buying patterns. Community Clustering is similar to Influences, but does not have a target variable.

Measure set types have associated roles, which may be either **Count**, **TargetSum**, or **SumSquared**. All measure sets contain a **Count** role measure. (Classification and Clustering measure sets must contain only the **Count** measure role.) Measure sets of the Regression type must contain the **Count** and **TargetSum** role measure, and optionally, the **SumSquared** role measure.

The **Count** role is a measure that counts the number of rows in the primary dimension associated with the measure set, such as the count of the rows in the individual dimension. The **TargetSum** role is a measure that is the sum of the quantity you are trying to predict when using this measure set. For example, if you want to predict the total amount that customers purchased, the **TargetSum** role is a measure that is the sum of the purchases made by customers. Thus, if the value you want to predict is x , then the **TargetSum** role is a measure whose value is $\text{SUM}(x)$.

When the E.piphany system builds models that need to compute statistical variance, it computes the sum of the squares of the value you are trying to predict for customers. For example, if the target value you are trying to predict is x , then the **SumSquared** role is a measure whose value is $\text{SUM}(x * x)$. Instructions for specifying the optional **SumSquared** role are given below.

A Classification measure set is always associated with an attribute. If that attribute is a List Membership attribute, then lists are used with the measure set. Attributes with dimension roles and dimension columns (regular attributes) specify the target attribute. Attributes are not associated with regression and clustering measure sets.

To define a measure set for classification:

1. Right-click the **Measure Sets** folder and select **New Measure Set**.

FIGURE 81: MEASURE SET DIALOG BOX

New Measure Set: IndividualCount

General Usage

Measure Sets define the target variables to predict.

Name: IndividualCount

Label: IndividualCount

Measure Description:

Type: Regression

Count: IndividualCount

TargetSum: Count of Individuals

SumSquared:

OK Cancel Apply

2. In the Measure Sets dialog box (Figure 81), enter the name of the measure set.
3. Enter its label (this is the name that end users see on Web pages).
4. When defining a measure set for a list membership target (that is, a measure set with no attribute and only the Count role defined), you can enter a short description in the **Measure Description** text box that shows up in the Influences Web page target selection box. This is useful for cases in which the list membership target is weighted by a measure that is not the count. For example, if the user associates Revenue with the Count role, then he or she may want to enter “weighted by revenue” in the description box of the measure set. Then, the target shows up in the Influences Web page as Member of List, weighted by revenue.

5. Select **Classification** as the measure set type.
6. In the **Count** drop-down list, select a measure that gives a count of the number of rows in the primary dimension. In some cases, you may wish to weight the rows in the primary dimension (for example, based on the amount of revenue from a customer). In such cases, the **Count** role should be assigned a measure such as a simple count of the units shipped, or a measure that sums the price of all products shipped, which produces a count that is weighted by price.

To define a measure set for the Community Clusters Web page:

- Select **Clustering** as the measure set type.
- Specify a **Count** measure role, which can be the count of the number of rows in the primary dimension, or possibly a weighted count (as described above for **Classification** measure sets).

Defining a measure set for regression is similar to defining one for classification:

- Select a measure for the **Count** measure role that counts the number of members of the primary dimension.
- Select a measure for the **TargetSum** measure role that is the sum of the fact column that corresponds to the fact value that you are trying to predict.
- In some cases, you may need to select a measure for the **SumSquared** measure role.

Usually, **SumSquared** can be computed from the **TargetSum** role, since the sum value corresponding to each row in the primary dimension can simply be squared. If the rows of the primary dimension table with which the measure set is used represent actual individual or households, you do not need to specify the **SumSquared** role. The **TargetSum** role computes the total amount of the target measure for each individual or household. That is, it computes the value x for each individual or customer, and this value can be squared for each individual or household. The E.piphany system can square this value and then sum it to obtain the right result.

If the rows in your primary dimension table represent some aggregate of individuals or households (for example, customer types or household types), then specify the **SumSquared** role. The **TargetSum** *measure* contains the sum of the target value for each customer type. The **TargetSum** *role* gives you the target value for each customer *type* (a sum over the individual customers), not the target value for each customer. Squaring the **TargetSum** role for the customer type does not square the values of the individuals aggregated to produce the customer type. Instead, it squares the value for the customer type aggregate, which is incorrect.

In such cases, at extraction time, you need to set up a fact table that contains the squares of the target value for each individual customer, and then sum these squared values to produce the sum-squared values for the aggregate customer types. You can then use this fact table to define a measure for the **SumSquared** role that, for each customer type, gives the sum of the squares of the individual customer target values.

PRESENTATION

The **Presentation** folder includes sub-folders for **Glossary Entries**, **Attributes**, **Web Pages**, and **Topics**.

GLOSSARY ENTRIES

Glossary entries define elements on a Web page to end users. To set up a glossary entry:

1. Double-click the **Glossary Entries** folder, which displays the Glossary dialog box (Figure 82).
2. Enter the name of the glossary term.

3. In the **Help Text** box, enter the definition that you want end users to see. This description can include HTML coding for links. For example: This is the dollar amount of revenue recognized. For more details, see our policy at:

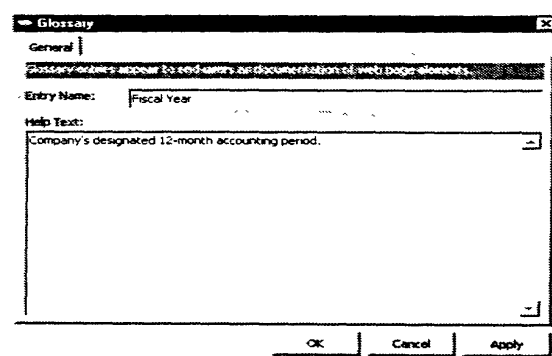
```
<a href="www.company.com/policies/revrec.html">
revenue recognition</a>
```

The entry is added to the **Glossary Entries** folder.

4. To create additional entries, right-click the **Glossary Entries** folder, and select **New Entry**.

You can also define a new glossary entry within the Measure Layout and Attribute dialog boxes.

FIGURE 82: GLOSSARY DIALOG BOX



ATTRIBUTES

Attributes, which are derived from columns in dimension tables, appear as items that end users can select on Web pages. Attributes also serve as filters that users apply to results to refine (drill down) their queries. The only difference between an attribute in a table and one in the E.piphany system is that an E.piphany attribute has an associated display label, such as Fiscal Year, that can be configured via EpiCenter Manager. Within an EpiCenter, an attribute is a global object; however, only attributes (and attribute roles) appropriate to a specific dialog box are displayed.

Note: *The `date_attributes.mdb` export file in the `ConfigFiles` directory contains common attributes based on the date dimension. If you import this file, you do not need to create attributes for fiscal year, month, and year name, and so forth. The filters for these attributes are dynamic—the check boxes/list boxes are based on the contents of your date dimension.*

To define an attribute, follow these steps:

1. Right-click the **Attributes** folder and select **New Attribute**.
2. In the Attribute dialog box (Figure 83), enter the name of the attribute and its label (the name that appears on the Web page).
3. Verify that the **Label Plural** is the correct plural of the label name (for multiple attributes).

4. Enter an abbreviation that the system can apply should the label name be too long (for example, **ASP** for Average Sales Price).

FIGURE 83: ATTRIBUTE DIALOG BOX: GENERAL TAB

Attribute Dialog Box: General Tab

General | Filter Elements | Usage

Name: Glossary Entry:

Label:

Plural Label:

Abbreviation:

Hyperlink:

Dimension Column:

Sort By:

Filter Type:

Number of columns:

Description:

5. Enter an optional hyperlink format in the Hyperlink text box. When you do, each attribute value that appears in a Web pages is displayed as a live hypertext link. You can use the string `val` to indicate where the attribute value is to appear in the URL for the link; for example:

`www.company.com/data/val`

6. Select the dimension column for this attribute from the base dimension table pane. Click a plus sign to expand the tree.
7. If you have an alternate column within the dimension role that provides an appropriate sorting order, you can specify that column in the **Sort By** drop-down list. For example, assume that your attribute is a date column that stores dates as text strings of the form:

`Month dy, year`

In this case, it might make sense to display the attributes in calendar order rather than by month. Choose **(default)** for the native sorting order of the attribute itself.

8. Select the attribute's filter type from the **Filter Type** drop-down list. Then use the **Select Attribute Values** pop-up menu to choose its display format:

- **Check Boxes**

Check boxes are rectangles next to an item that a user points and clicks to check or uncheck. Users may check more than one item in a column.

For the display format, select the number of columns for the attribute values from **Number of columns**.

- **Dynamic Check Boxes**

Dynamic check boxes are refreshed each time the AppServer starts up; static check boxes never change. For the display format, select the number of columns.

*Note: To generate check boxes and dynamic check boxes, you need to enter the proper SQL. Clicking the **Edit SQL Query** button in the **General** tab of the **Attribute** dialog box displays a template that you can modify for this SQL.*

- **Dynamic Listbox**

A dynamic list box contains a list of items for filter selection that are refreshed each time the AppServer starts up. The end user clicks the down arrow to the left of the box to display a drop-list menu of all list items.

For the display format, select the height of this drop-down list from **List box height**.

- **Entire Dimension**

All attributes of the dimension are available as a filter. (This filter type is for special purposes in some data-mining algorithms.)

- **List Membership**

This drop-down contains a list of lists. These lists are created by end users and stored in the Report Gallery.

For the display format, select the height of the drop-down list.

- **Radio Buttons**

Radio buttons allow the end user to make one choice only. For the display format, select the number of columns.

- **Listbox**

A list box contains a list of items for selection. The end user clicks the down arrow to the left of the box to display a drop-down of the items in the list.

For the display format, select the appropriate height for this drop-down list.

- **Text Box**

A text box is a blank area in which the user may enter text.

For the display format, select the height of the text box from the counter labeled **Text box height**.

Note: The primary dimension attribute for an Influences Web page must have the filter type of Entire Dimension or List Membership, which is available for indiv or group only.

9. As with measurements, attributes can be defined in the glossary. You can select an existing glossary entry from the drop-down list, or create a new one.

To set up a glossary entry, click **New** and type the entry name and help text (that displays to end users) in the Glossary dialog box (see Figure 82). The **Help Text** description you enter can include HTML coding for links.

FILTER ELEMENTS AND FILTER GROUPS

Filters allow Web page users to restrict the data accessed for a query to specific attribute values (values in dimension columns); for example, the data can be filtered so only the values for direct sales during the years 1990 through 1995 are returned. Filter elements are specific dimension-column values that users can choose. A *filter element* appears on the Filter Pop-up as a single check box or entry within a list box. A *filter group* is a logical grouping of filter elements. For example, a filter by year might have the a filter group of Q197 (first quarter of 1997), that includes the months January '97, February '97, and March '97.

Filter elements can be assigned to groups dynamically or statically. For dimension columns that contain changing values, it is often better to choose a dynamic filter type. For columns that seldom change, a static filter type (one for which the label in the General tab does not include the word “dynamic”) is often preferred.

DYNAMIC FILTER GROUPS

When you choose a dynamic filter type, **Edit SQL Query** on the General tab is activated. Click this button to display the Fill from Query dialog box, which you can use to prepare an SQL query to populate the filter groups for this attribute. The AppServer invokes this query whenever it starts up.

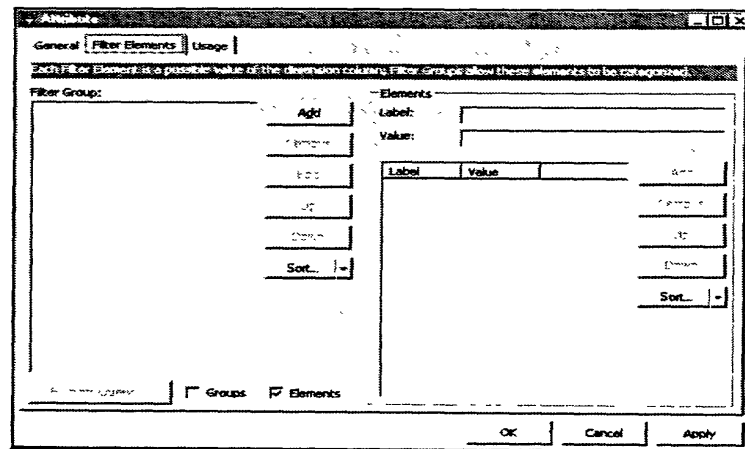
1. Click the **Template** button in the Fill from Query dialog box to display a query template. If you do not want to classify the filter elements into groups, you can use the template query as it is. If you do want to classify filter elements into groups, you can replace the **ALL** token with an expression or create a union of **SELECT** statements with different **WHERE** clauses. Click **OK**.
2. To preview the filter groups produced by your query, select the Filter Elements tab and click **Show Query Results**.

STATIC FILTER GROUPS

When you choose a static filter type, you can use the Filter Elements tab to define filter groups and assign elements to them. Follow these steps to create filter groups for static filter types:

1. Click the **Add** button in the Filter Elements tab to create a new filter group, then enter a label for that group in the Filter Group dialog box. You can also add a glossary entry for the new filter group. (See “Glossary Entries,” on page 262.)
2. Continue adding new filter groups. When you are done, you can specify a sort order for the groups as they are to appear on the Filter pop-up Web page by clicking **Sort** and choosing an option from the pop-up menu. You can also move a label to a different place in the list by choosing it and clicking the **Up** or **Down** button.

FIGURE 84: ATTRIBUTE DIALOG BOX: FILTER ELEMENTS TAB



3. Choose a filter group from the list of filter groups that you just created.
4. You can add filter groups and elements directly by entering attribute values and labels in the **Elements** pane, or you can create a list of elements with an SQL query.

5. To create a list of elements with an SQL query, click the **Groups** and **Elements** check boxes, then click **Fill from Query**.
6. In the Fill From Query dialog box, click **Template** to display an SQL template that you can use to populate the filter group.
7. Edit the SQL template to restrict the column values to only those values you want to include in the current filter group.
8. Click **Test** to verify the query, then click **OK** to assign column values to filter elements in the current filter group. Be sure to append the token `_0$$CURR` to any table names that you add or replace; for example, `Product_0$$CURR`.
9. Update the labels for filter elements. Enter the label for an element in the **Elements** pane, then click the label that you want to update in the **Label** column. The label updates automatically.

TRANSACTION FILTERS

Transaction filters, which are specific to the List Manager, enable users to filter group and indiv dimensions by their participation in a fact, such as the purchase of a product. Data related to facts, such as *how much* of an item people bought, is a numeric value, or a measure. To enable end users to apply a filter on measure data in this manner, you need to define transaction filters (via the Web page dialog box) in addition to regular filters. See Chapter 4, “Configuring Web Pages and Topics,” for instructions.

ATTRIBUTE FACTORY: FILTERS FOR SURVEYS

For survey purposes, more than one filter on the same attribute may be desired. When extracting schema for survey data, you can use the Attribute Factory dialog box (accessible from a Transaction Filter dialog box) to create transaction filter filters, or two-occurrence filtering. (Since filters are really attributes, you are creating a new attribute, hence the term *Attribute Factory*.)

For example, assume you have an individual base dimension with an **indiv** dimension role that points to a fact table named **Answered**. The **Answered** fact table, in turn, points to the **Answer** dimension table that contains three columns: **Question**, **Answer**, and **Question and Answer Concatenated**. The **Question** column lists each possible question. The **Answer** column lists each answer to each question. The **Question and Answer Concatenated** column shows the distinct combination of each question/answer pair.

TABLE 9: ANSWER DIMENSION TABLE

Question	Answer	Q and A Concatenated
Favorite Color	Red	FavCol:Red
Favorite Color	Blue	FavCol:Blu
Favorite Color	Blue	FavCol:Blu
Favorite Color	Green	FavCol:Gre
Favorite Color	Red	FavCol:Red
Political Affiliation	Republican	PolAff:Rep
Political Affiliation	Democrat	PolAff:Dem
Political Affiliation	Republican	PolAff:Rep
Political Affiliation	Democrat	PolAff:Dem
Political Affiliation	Independent	PolAff:Ind

Assume that an Individual Campaigns Web page has demographic filters for City and Income. The end user wants to further drill down to find out which individuals responded in a certain way to one or more questions, such as those individuals from Omaha with an income in the \$50,000 range who are not affiliated with one of the two major political parties and whose favorite color is red (by applying two transaction filter filters).

To enable this type of two-occurrence filtering, design the Web page with a transaction filter named, for example, **Answered**, that consists of the unique question-answer pairs as transaction filter filters. (**Answered** represents a standard transactional filter.)

In this example, the check or list boxes available from the **Answered** transaction filter drop-down list derive from the **Question** and **Answer Concatenated** column of the **Answer** base dimension table. The query machinery needs to be able to uniquely identify a row; one way to do this is to concatenate question and answer values.

Follow these steps to create multiple transaction filter filters:

1. Open a Transaction Filter dialog box and click the Transaction Filter Filters tab. Click the **Factory** button to open the Attribute Factory dialog box (Figure 85).
2. Select the unique question/answer pair dimension column to be filtered. (**Question** and **Answer Concatenated** in the example).
3. Select one of these filter types: list box or check box.
4. Click **Template** and fill out the query template.

The `attribute_name` must be a unique name that is tied to a question. For `<YOUR_EXPRESSION>`, enter `['Question' $$CAT question]` where *question* is the name of the **Question** column in the **Answer** base dimension table. (The concatenation ensures that the names do not clash with other attributes.) There should be one `attribute_name` per question.

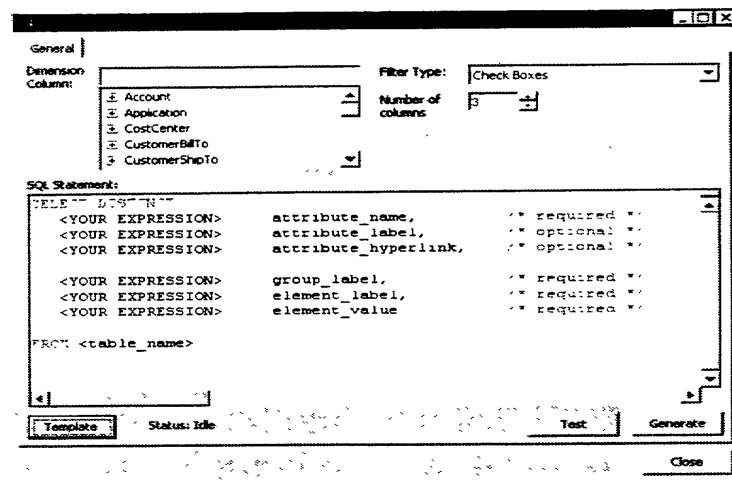
If appropriate, assign values to `attribute_label`; for example, `attribute_label = question`, and to `attribute_hyperlink`.

The `filter_group_label` is a further grouping mechanism. It is only for subcategories, for example, if one wanted to group Political Affiliation by categories such as frequent or infrequent voters. It can also be a literal, such as **All**. (A filter group label applies only to a check-box filter type.)

For the `filter_element_label`, enter the name of the **Answer** column in the **Answer** dimension table. These are the labels of the individual columns in the **Answer** drop-down list on the Web page.

For the filter `element_value`, enter the name of the unique question/answer pair column.

FIGURE 85: ATTRIBUTE FACTORY DIALOG BOX



SECURITY

The Epiphany system provides two areas of security:

- *Authentication*, or a user's ability to log on to the system. Authentication is determined by the Windows NT operating system.
- *Access rights*, or the permissions a user has after logging on. Access rights include the user's ability—
 - to open a Web page associated with a topic/navigation node.
 - to perform a specific navigation step.
 - to save queries for those Web pages. Saved queries (called *reports*) are lists of the options that a user selected on a Web page (to generate a report).

- to access data based on the values of dimensional attributes; that is, to restrict access rights to the Web page to certain dimensional attribute values. You can use this kind of security to allow some users to see data for one region but not another.

The E.piphany system can also use Windows NT groups to administer access rights.

You can use the **Security/Storage** folder in the EpiCenter Manager directory tree to add groups and users to the system. The two work in tandem: groups have users, and users belong to groups. When setting up a new E.piphany system, first set up groups and then add members to them after defining users. (Windows NT groups members can simply be imported.)

The **Security/Storage** folder also contains the Report Gallery, which enables the administrator to organize all saved reports in folders for groups and users. See “Report Gallery,” on page 287 for more information.

A group can be marked administrative. Users that are members in this group are administrative users that have these special privileges:

- access to all folders and reports
- all navigation paths are enabled
- all column restrictions are disabled
- monitor functionality is exposed to administrative users only

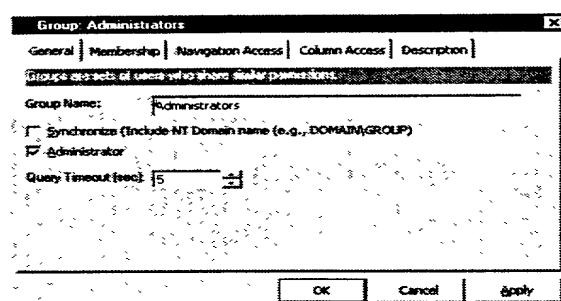
SETTING UP GROUPS

Members of a group share similar access rights, or permissions. In general, you should attach these to groups instead of users to simplify maintenance. Access rights set for a user, however, take precedence over access rights set for groups to which the user belongs.

To define a group, right-click the **Group** folder and select **New Group** from the pop-up menu. (This menu also has commands for exporting all groups, importing NT Groups, and refreshing the group folder list.) Complete the information requested in the Group dialog box tabs (General, Membership, Navigation Access, Column Access, and Description) as described in this section. (Use the Description tab to document the group for your own reference.)

1. In the General tab (Figure 86), enter the group's name.

FIGURE 86: GROUP DIALOG BOX: GENERAL TAB

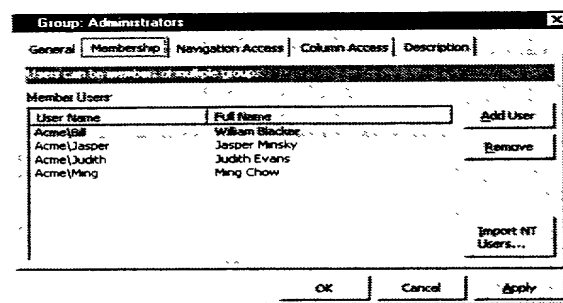


2. Select the **Synchronize** option to add new users to the group in an “autopilot” fashion via Windows NT synchronized groups. You need to precede Windows NT synchronized group names with their Windows NT domain name prefix (and matching name).

Each time a user logs in, the Windows NT security API is accessed for a list of group names to which that user belongs. If the user is a member of an NT group that also exists in the Epiphany world and is marked synchronized, and there is no membership record for that user and that group in EpiCenter Manager, then a new record is created. Conversely, if the user is no longer a member of an NT group, but there is such a membership record in EpiCenter Manager, and the group is marked synchronized, then the latter record is removed.

3. Select **Administrator** if members of this group have administrative rights. An Administrator can see all Web pages and reports in the system, but cannot modify special folders, such as the **Public** folder. (See “Report Gallery,” on page 287 for a discussion of special folders.)
4. Enter the seconds for the maximum time that a query should run for members of this group. As an administrator, you may set a time-out limit to ensure that someone does not monopolize database engine resources.
5. You can use the Membership tab (Figure 87) to add already created users to the group (as described in “Setting Up Users,” on page 279). Users may be members of multiple groups. Click **Add User**.

FIGURE 87: GROUP DIALOG BOX: MEMBERSHIP TAB

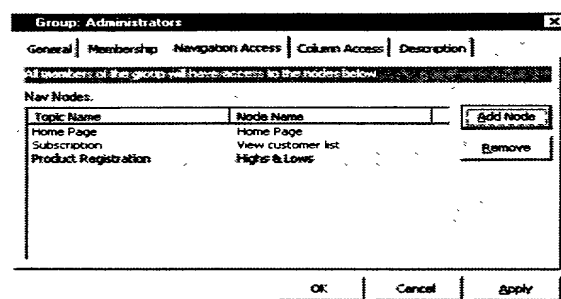


6. Select the user's name from the Choose User dialog box and click **OK**, or click **New**, which displays the User dialog box.
7. To import NT users into the group, click **Import NT Users** and enter the domain in the dialog box. You have the option of adding the new users to all E.piphany groups in which they are members of corresponding NT groups. This actions adds users to existing groups; it does not add new E.piphany groups.

8. You can use the Navigation Access tab to assign all members of the group access to topics and navigation nodes within topics. Before E.piphany users can open a Web page, they must be granted access to its associated topic/node. Users have the ability to access all nodes granted access to their groups by the administrator. Click **Add Nodes** and select the topic/node from the list.

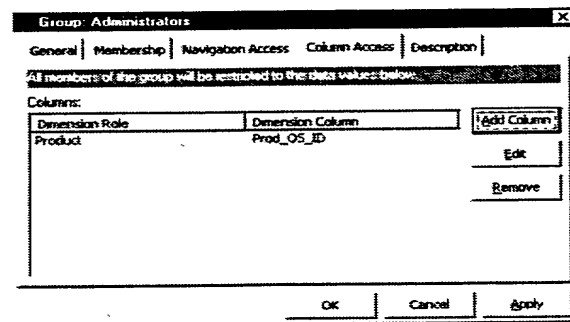
*Note: Before you can configure solutions or create new topics, permissions must be set for all nodes. Otherwise, these nodes do not appear when you start the AppServer. You can select all nodes when adding them to a user or group. Alternatively, you can open the topic, select the Navigation Nodes tab and multi-select all of the nodes in the list. Click **Add Group**, and choose the group to assign all members of the group permission to see all of the nodes in the topic.*

FIGURE 88: GROUP DIALOG BOX: NAVIGATION ACCESS TAB



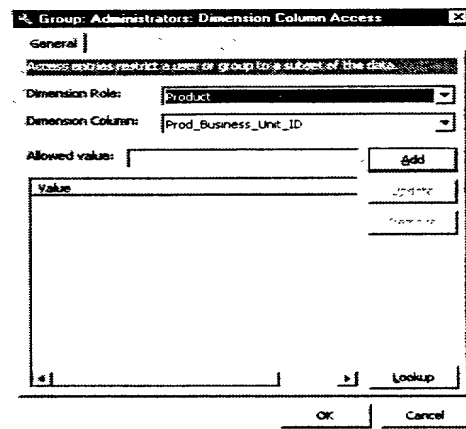
You can use the Column Access tab (Figure 89, on page 278), which is the same for users and groups, to restrict access rights for the Web page to certain attributes (dimension columns).

FIGURE 89: GROUP DIALOG BOX: COLUMN ACCESS TAB



1. Click the **Add Column** button, which displays the Dimension Column Access dialog box (Figure 90).

FIGURE 90: DIMENSION COLUMN ACCESS DIALOG BOX



2. Select the Dimension Role for this dimension column from the drop-down list.

09625518.072500

3. To display all of the values in the dimension column, click **Lookup**, which directly accesses the EpiMart data. Select a value or values from the listing and click **OK** to add it to the value list.

The values entered in this list should correspond to actual database values in the base dimension table to which that dimension column corresponds. For example, selecting `Date.fy_name` as the field with the values 1997 and 1998 causes all reports to be filtered with these values.

4. If you know the exact value of the dimension column whose values are accessible to the group, type it in the **Allowed values** text box, and click **Add** to place it in the **Value** listing below. Repeat this step to add additional values.
5. Click **OK**.

After a group has been defined, it becomes a sub-folder of the **Group** folder. You can right-click a group folder and use the pop-up menu to set up a new group, or to edit, delete, or duplicate the group. A duplicate group is identical except for its group name.

SETTING UP USERS

All authorized users of the E.piphany system appear as icons in the **Users** sub-folder of the **Security/Storage** folder. By default, new users have these permissions:

- no access to any topics/nodes.
- no group membership.
- inability to save any reports.
- access to all data in all dimensions—there are no dimensional attribute (column) restrictions.

For column restrictions only, the access rights set for a user have precedence over access rights set for groups to which the user belongs.

To set up a new user, right-click the **Security/Storage** folder and select **New User**. The User dialog box (Figure 91) that displays has tabs for General, Membership, Navigational Access, and Column Access.

Configure the General tab as follows:

1. Enter the person's username and first and last names.

The first and last names are used for display purposes only. If they are missing, then the username is displayed.

For Windows NT authentication, enter users with their Windows NT domain prefix to distinguish among users with the same name but who are in different domains.

2. Select the Report Gallery type (view mode) for this user from the drop-down list.

There are three Report Gallery modes:

- **FoldersAndFiles**, which shows both folders and files. It allows you to switch to **Files** mode.
- **Files**, which shows all files that the user has access to. It allows you to switch to **FoldersAndFiles** mode.
- **FoldersAndFilesOnly**, which shows both folders and files. You cannot switch out of this mode.

3. Check whether the user should go directly to the results of a report for gallery, mailed, and favorite reports. If **executed is not checked** for one of these options, the user is sent to the query parameters on the Web page instead of the report itself (and must generate the results).

Note: A user can e-mail another user a report by typing the URL in the text, which the recipient can then click to open the report in a browser window.

4. Indicate the maximum number of favorite reports that this user may have.

The user can choose how many favorite reports appear below each topic on a home page. Although there is no maximum number, five reports per topics is recommended.

5. Enter the seconds for the maximum time that a query should run for this user. As an administrator, you may set a time-out limit to ensure that a user does not monopolize database engine resources.

FIGURE 91: USER DIALOG BOX: GENERAL TAB

User: Acme\Ming

General | Membership | Navigation Access | Column Access

A new record represents a distinct named user of the Enterprise system.

Domain\User Name: Acme\Ming
(Include NT Domain name: e.g. DOMAIN\USER)

Full Name: Ming Chow

Report Gallery Type: FoldersAndFiles

☐ Execute gallery reports
☒ Execute mailed reports
☒ Execute favorite reports

of Favorite Reports: 5
 Query Timeout (sec):

OK Cancel Help

After groups have been created as described in “Setting Up Groups,” on page 274, you can use the Membership tab to assign group memberships to the user. Click **Add Group** and select one or more groups from the Choose Groups dialog box. (Hold down the Shift key to select more than one.)

A user can be a member of multiple groups, but for security reasons needs to be a member of a single group (called the primary group) when running queries. The restrictions set for the primary group apply to all of the user’s queries. After assigning the user group memberships, select the main group in the list and then select the **Primary** option in the tab.

Note: *Adding users to multiple groups makes it easier for users to share saved reports. A user may save reports into any group folder for which he or she is a member.*

If there is no group that contains all of the user’s privileges, or if the user has privileges that apply only to that user and no other, assign privileges to the user. Individual privileges that you set always override group privileges, even if the user is a member of a primary group.

Before E.piphany users can open any Web page, they must be granted access to its topic and node. Users have the ability to see and use all nodes to which they have access, and to all nodes granted access to their groups. In the Navigational Access tab, click **Add Nodes** and select the Nodes in the list that the user may access. Hold down the **Shift** key to make multiple selections.

You can use the Column Access tab to restrict access rights for the Web page to certain attributes (dimension columns). Follow these steps:

1. Click **Add Column**, which opens the Dimension Column Access dialog box (Figure 90, on page 278).
2. Select the Dimension Role for this dimension column from the drop-down list.
3. To display all of the values in the dimension column, click **Lookup**, which directly accesses the EpiMart data. Select a value or values from the listing and click **OK** to add it to the value list.

The values entered in this list should correspond to actual database values in the base dimension table to which that dimension column corresponds. For example, selecting `Date.fy_name` as the field with the values 1997 and 1998 causes all reports to be filtered with these values.

4. If you know the exact value of the dimension column whose values are accessible to the group, type it in the **Allowed values** text box, and click **Add** to place it in the **Value** listing below. Repeat this step to add additional values.

STORAGE

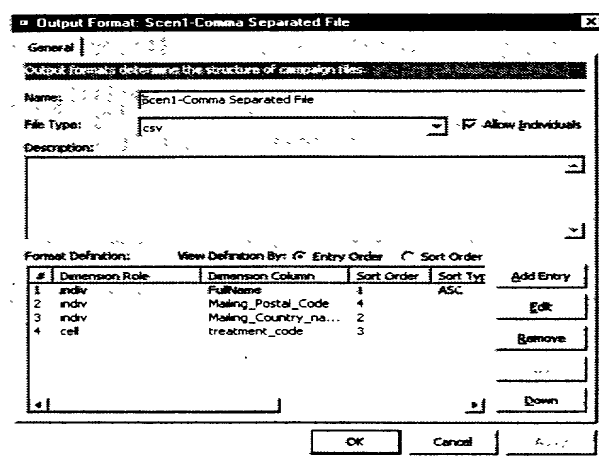
In addition to the security settings folders, the **Security/Storage** folder also contains folders for campaign output and report configuration.

OUTPUT FORMATS

An output format specifies the structure of the files generated in a campaign. To define a new output format:

1. Right-click the **Output Formats** folder and select **New Output Format**.
2. In the **Output Format** dialog box (Figure 92), enter a name and a description for the output format.

FIGURE 92: THE OUTPUT FORMAT DIALOG BOX



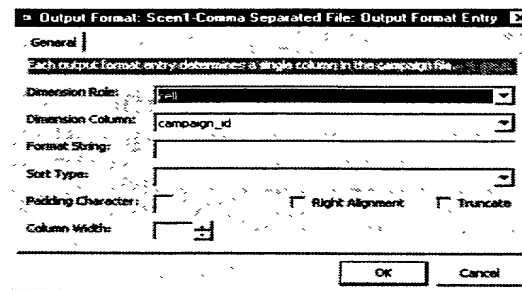
3. Choose an output file type from the drop-down list. The following file types are available:
 - **csv** — Comma-separated values (text format)
 - **html** — HTML
 - **tsv** — Tab-separated values (text format)

- **txt** — Text (text format)
 - **xls** — Excel spreadsheet
4. Check **Allow Individuals** if this format should allow attributes for both individuals and groups. (If this is not checked, then individual attributes cannot be specified in the format.)
 5. Add the entries to be included in the output.

To add an entry:

1. Click **Add Entry**, which displays the **Output Format Entry** dialog box (Figure 93).

FIGURE 93: THE OUTPUT FORMAT ENTRY DIALOG BOX



2. If the entry is a dimension column value, select the dimension and column from the drop-down list. If this does not apply, you can use the **Format String** text box.
3. In the **Format String** text box, if the entry is text format (**txt**, **csv** or **tsv**), enter a standard `java.text.MessageFormat` formatting string. See your Java Development Kit (JDK 1.1) documentation for instructions. The URL is:

[http:// java.sun.com/products/jdk/1.1/docs/api/
java.text.MessageFormat.html](http://java.sun.com/products/jdk/1.1/docs/api/java.text.MessageFormat.html)

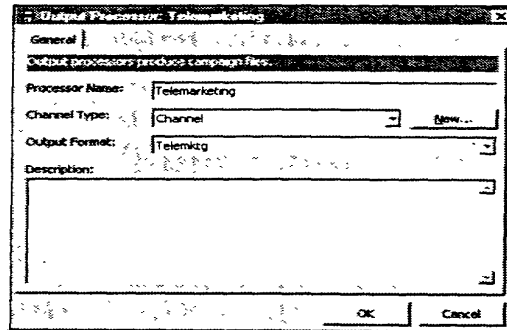
For example, the format string `Year {0}` translates to the value of `Year 1997` when the value of the dimension column in the database is 1997. The format `{0,number,percent}` translates to the value of `35.5%` when the value of the dimension column is 0.355.

4. For `html` or `xls` file types, enter the name of the column header without quotes. The backslash (`\`) character is special, as long as it is followed by either `n`, `r`, or `t`; as in `\%` (where `%=n,r,t`). If the backslash is not followed by one of these three characters, it is ignored. A double backslash (`\\`) is interpreted as a single (`\`).
5. Select one of the sort types from the drop-down list.
If nothing is set, then the records in the file are not sorted. If `ASC` or `DESC` is selected, then records are sorted in ascending or descending alphabetic order, respectively. `EXCLUDE_ASC` or `EXCLUDE_DESC` are used for sorting records internally, not for sorting output.
6. For text formats, enter any padding characters to be added to the entry record. These characters fill out the column to the specified column-width.
7. Check **Right Alignment** if you want the entry to be padded to the left of the record. If unchecked, padding occurs to the right.
8. Check **Truncate** if you want the field to be truncated if it is greater than the column width.
9. For text formats only, select column-width options. A column width of 0 indicates an arbitrary column width.
10. Click **OK** to add the entry and return to the **Output Format** dialog box.
11. If multiple fields are to be sorted, select **Sort Order in View Definitions By**. Move entries to the appropriate location using the **Up** and **Down** buttons.
12. Select **Entry Order in View Definitions By** to view the order in which the entries appear in the output.

OUTPUT PROCESSORS

An output processor specifies an output option that is available to the end user. Output processors make use of output formats, so you need to define an output format in order to define an output processor.

FIGURE 94: THE OUTPUT PROCESSOR DIALOG BOX



To define an Output Processor:

1. Right-click the **Output Processors** icon and choose **New Output Processor**.
2. In the **Output Processor dialog box** (Figure 94), enter a name and description for the output processor.
3. Choose a channel type from the drop-down list. Use the **New** button to define a new channel.

A channel is a category of processors grouped by the type of campaigns that they execute; for example, Direct Mail, Telemarketing, Web Site, and so forth.

4. Choose an output format from the drop-down list.
5. Click **OK** to define the output processor.

REPORT GALLERY

Note: The Report Gallery in EpiCenter Manager is for administrative use only. Many of the features of this Report Gallery can also be accomplished by end users using the Report Gallery in the Web interface.

You can use EpiCenter Manager to browse the saved reports for a Web page, as well as view who has access to those reports. All of the saved reports for the EpiPhany system are organized in folders and saved reports in the Report Gallery. When a user saves a report, it is saved in whatever folder the user is currently in. This can be any folder to which the user has write access.

The Report Gallery has a top-level folder that contains the **Public** folder, **All Users** folder, and **All Groups** folders. Everyone has access to **Public** folder by default. Any user can save reports there. There is a folder for each user under the **All Users** folder. There is also a folder for each group under the **All Groups** folder. Every user folder, group folder, and public folder has three sub-folders: **Defaults**, **Favorite Reports**, and **Favorite Charts**.

A default report is the set of Web page settings that a user sees when he or she first opens that Web page. In general, the most specific default report is used. Thus if a user-level default report exists, that report is opened in the user's Web browser. If no user default report exists but a group default report exists for the Web page, the group default report is displayed. The default report in the **Public** folder should always be available if there is no specific user or group default report. Whenever a new Web page is created, the administrator should save one report in the **Default** sub-folder in the Report Gallery's **Public** folder and make it the default.

When a report is saved in a **Favorite Reports** folder, it shows up on the home page under the topic for that report.

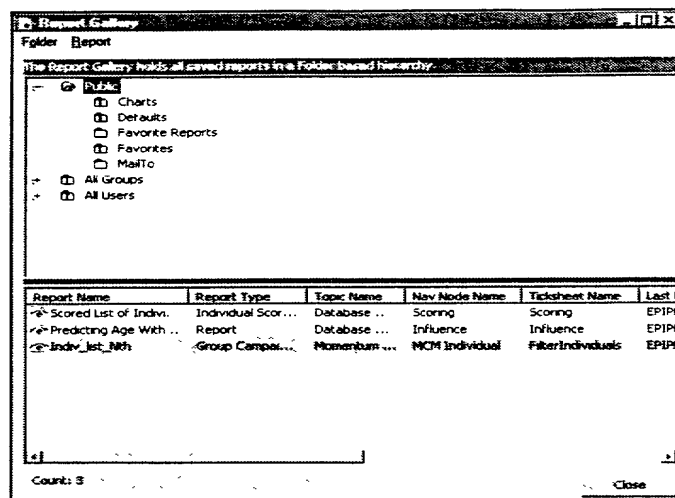
If you enlarge a chart on the results page, you can save it in the **Favorite Charts** folder, which shows the chart on the home page. (The chart is also scheduled for updating in the Charts queue.)

Using EpiCenter Manager, the administrator can control the access to the Report Gallery's folders. For example, modifying the access to the public **Defaults** folder determines who can save public defaults (by default everybody can save public defaults). Turning off all permissions for the **Public** folder disables Public folder completely. Turning off permissions for the **All Groups** folder disables the **All Groups** folder completely. Do not remove the folders. Modify permissions on them if you want to control the access to them.

***Note:** When you change the names of attributes, transaction filters, output processors, topics, navigation nodes, options, measures, measure sets, and Web pages, you are requested to confirm the name change. These are unique name fields in metadata objects that saved reports reference internally. If these names are changed, then parts of the saved report may become invalid. For example, if an attribute has been renamed, and you open a saved report that uses that attribute and attempt to execute it, you receive an error message because the attribute is invalid. If, however, you open the saved report without executing it, the Web page has the default attribute. You can simply re-save the report with the new attribute.*

To display the Report Gallery, right-click the **Security/Storage** folder and select **Report Gallery** from the pop-up menu. The Report Gallery (see Figure 95) is displayed. The top pane shows the folders organized in a tree hierarchy, and the lower pane lists the reports for a selected folder by report name, report type, topic name, navigation node name, Web page name, date last modified, and the person who modified it.

FIGURE 95: REPORT GALLERY IN EPICENTER MANAGER



The Report Gallery's main menu has menus for **Folder** and **Report**, which are described below.

FOLDER MENU

Note: *Right-clicking in the top pane of the Report Gallery displays a pop-up menu with the Folder commands.*

You can use the **Folder** menu to:

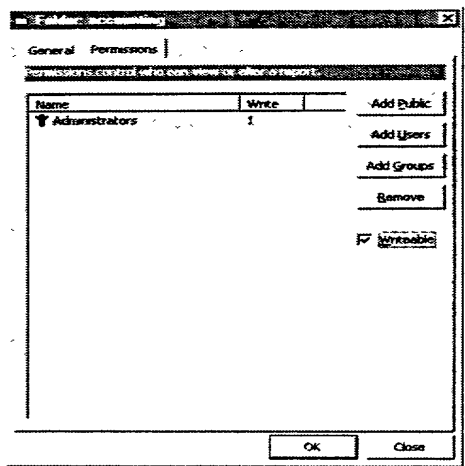
- create a folder for saved reports. Choose **New** from the **Folder** menu, select the folder or sub-folder, and enter the new folder name in the dialog box.
- delete a selected folder.
- display the folder's properties. (You can also double-click the leaf folders to open the Folder dialog box.)

The Folder dialog box has two tabs: **General** and **Permissions**. The **General** tab shows the folder's name, path, date last modified, and any description.

The Permissions tab allows you to set controls for who can view and alter the folder. See “Setting Permissions,” on page 292 for more information. *This is the only way to modify public defaults.*

- find a report in one of the folders.
Enter the file name in the Find Report dialog box. You can refine your selection to report types and date modification ranges. Pressing the **F3** key locates the next report that meets this criteria.
- move folders by dragging them.

FIGURE 96: FOLDER DIALOG BOX: PERMISSIONS TAB



REPORT MENU

Note: *Selecting a report file in the lower pane displays a pop-up menu with the Report commands.*

You can use the **Report** menu to copy and move reports and folders. (This functionality is not available through the Web interface Report Gallery.)

To copy a report:

Select it, and choose **Copy** from the **Report** menu. Select the folder in which you would like to place the copy of the report, and select **Paste**.

You can also cut and paste a report, or explicitly delete it.

Double-clicking a report opens the Report dialog box (Figure 97).

- The General tab shows the report's name, folder path, report type, Web page name, date last modified, and any description.
- The Permissions tab allows you to set controls for who can view and alter the report. See *Setting Permissions* for more information.

When a user saves a report without explicitly setting permission, default permissions are assigned. See "User Default Permissions" on page 292.

FIGURE 97: REPORT DIALOG BOX

The screenshot shows a dialog box titled "Report: Scored List of Individuals". It has two tabs: "General" (selected) and "Permissions". The "General" tab contains the following fields:

- Report Name:** Scored List of Individuals
- Folder Path:** Public
- Report Type:** Individual Scored List
- Last Modified:** EPIPHANY\brianb 3/12/99 7:02:00 PM
- New Node Name:** Database Marketing Scoring
- Web Page Name:** Scoring
- Description:** (empty text area)

At the bottom right of the dialog box are "OK" and "Close" buttons.

SETTING PERMISSIONS

You can use the Permissions tab of the Folder and Report dialog boxes (see Figure 96) to assign users and groups permission to view a folder or report.

If you select **Writable** in the Report Permissions tab before clicking the **Add** button, then the users or groups you add may alter the report. Selecting **Writable** in the Folder's Permissions tab gives write access to any report in the folder for the user or group (although a user cannot create a new report in a folder unless he or she has write access to that folder).

- To allow all users the ability to view this folder or report, click **Add Public**.
- To give users access to folders/reports, click **Add Users** and select users from the Choose dialog box.
- To give groups access to folders/reports, click **Add Groups** and select groups from the dialog box.

USER DEFAULT PERMISSIONS

When a user saves a report without explicitly setting permission, default permissions are assigned. Default permissions are calculated as follows:

- If the user creates a report in a regular folder, the folder's permissions are used.
- If the user creates a report in a user folder, default permissions on the report are Read/Write for the user only.
- If the user creates a report in a group folder, default permissions on the report are Read/Write for the group only.

By default, others have read permission on a user's folder. A user cannot set up his or her user folder (private folder) in such a way that other users can create, delete, or update properties on folders or reports in that folder. It is possible, however, for users to overwrite reports that they have access to in the user folder.

To create, delete, or update a report or folder, a user must have write permission to the folder in which the operation is about to be performed (the parent folder). Overwriting a report requires read-permission-only for the parent folder (not write permission).

GENERATING SCHEMA

After defining your metadata via EpiCenter Manager, you need to convert these definitions into actual tables (the EpiMart). You can use the Generate Schema dialog box to build the EpiMart tables, as well as to populate the physical date dimension table. The Date dimension table is a special base dimension table supplied by E.piphany for storing all attributes related to time. All fact tables in an EpiCenter receive a foreign key (called `date_key`) to this table.

To generate the schema for your EpiMart:

1. Choose **Generate Schema** from the **EpiCenter** menu. The Generating EpiMart Schema dialog box has two tabs: Schema and Results.
2. In the upper pane of the Schema tab, select **Trial Run** to see what the results are without making permanent changes to the files.
3. By default, only tables that have changed since the last time you generated schema are rebuilt. Select **Force Rebuild of all Tables** only if you want to rebuild *all* of your tables.
4. To generate schema for the Campaign Manager-related tables, check **Include Backfeed Tables**.
5. Check **Build Sampling Table**. The sampling table is a master table (a list of random numbers) that E.piphany programs use to produce samples of dimension tables. Select the number of rows to be populated in the sample dimension tables.

Set this value to *at least* the larger value of—

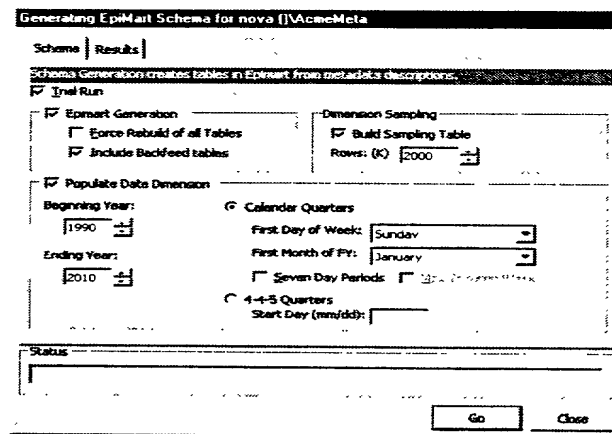
- the number of rows in the individual dimension, or
- the number of rows in the group dimension.

This value should be larger than the number of rows in the largest dimension table.

6. The date dimension table must be populated the first time you generate schema. Follow the instructions for populating the data dimension below.
7. Click **Go** (and watch the progress bar).
8. After the trial run is finished, click the **Results** tab to view which tables were updated.
9. If the results are acceptable, de-select **Trial Run** in the Schema tab, and click **Go**.

After the schema has been generated, the system records the current state of the metadata so that the next time the schema is updated, the current definitions can be compared with the new ones, and the appropriate tables can be created or altered as necessary.

FIGURE 98: GENERATING EPIMART SCHEMA DIALOG BOX



To populate the date dimension table:

1. Select **Populate Date Dimension** in the Schema tab of the Generating EpiMart Schema dialog box. (You can also choose **Populate Date Dimension** from the **EpiCenter** menu to populate the dates only; see “Populating the Date Dimension,” on page 151.)
2. Enter the values for the beginning and ending years of the EpiMart. The date range of the EpiMart should be at least as large as any dates that are found in the data you are extracting. These values are defined in the Configuration dialog box (choose **Configuration** from the **EpiCenter** menu to open it). For users to obtain the best results when forecasting trends, you need to build the date dimension as far into the future as you plan to forecast. (Currently, the maximum prediction is three years past the last date that has recorded data.) If the date dimension is not built out far enough, the user receives columns with names such as:

Dec 1999, Second Next, Third Next

instead of:

Dec 1999, Dec 2000, Dec 2001

3. Choose the appropriate quarter system: Calendar Quarters (three month divisions of the year) or Quarters 4-4-5. *4-4-5 Quarters* represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.
4. *For Calendar Quarters*, enter the first day of the week and the first month of the fiscal year. Select **Seven Day Periods** to ensure that all fields that count up in number (such as `week_number_fq`) begin with seven day period; for example:

11111112222222333333...13 or 14

whereas all fields that count down in number, such as `week_number_til_end_gq` end with seven day periods; for example:

14 or 13...333333222222111111

The alternative is that weeks may overlap quarter boundaries incompletely.

5. For **Calendar Quarters**, select **Max Thirteen Week** to prevent a “14th” week or “53rd” year; the 13th or 52nd weeks are simply extended as needed.
6. For **4-4-5 Quarters**, enter the start date of the first quarter.

Appendix C, “Date Dimension Fields,” describes the date dimension fields used by the E.piphany system.

EXPORTING/IMPORTING METADATA

E.piphany provides a metadata export/import utility for moving metadata between EpiCenters and for backing up the definition of an EpiCenter. To use this tool properly, you need to understand the metadata concepts (see Metadata Overview in Appendix G, “Export/Import of Metadata”).

Note: Within the EpiCenter tree in a single EpiCenter Manager window, you can drag and drop items or folders from one EpiCenter to another.

The export operation produces a Microsoft Access database that contains a representation of the metadata that was chosen for export. Upon import, this representation is converted back into valid E.piphany metadata in the target EpiMeta. Using EpiCenter Manager, you have granular control over which metadata objects get exported during each operation. For example, you can use the same objects and Web pages in several EpiCenters. Also, during import, you control whether or not to overwrite existing metadata that conflicts with what is in the import file.

Reasons for exporting files other than publishing them for import are to save files for document control (source safe) and to send files as an e-mail attachment; for example, you might e-mail an exported file to Technical Support for analysis.

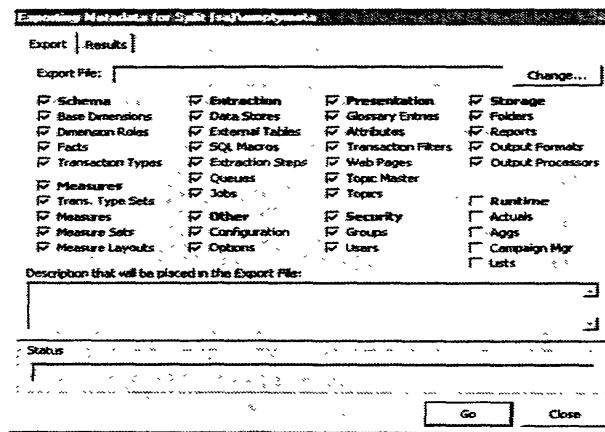
You can export individual metadata objects by right-clicking folders in the EpiCenter Manager directory tree and selecting the **Export** command from the pop-up menu. This option is available only for exportable metadata objects.

005520.072500

To display a dialog box in which you can export all or any subset of metadata files:

1. Right-click the EpiCenter icon and select **Export\Export All** from the pop-up menu, which displays the Exporting Metadata dialog box (Figure 99, on page 297).

FIGURE 99: EXPORTING METADATA DIALOG BOX



2. In the Export tab of the Exporting Metadata dialog box select a heading, such as **Schema** to select all of the Schema topics automatically, or select individual subtopics.
3. Enter any description to be placed in the export file. This description can be read in the Importing Metadata dialog box when the user selects this file for import.

4. Click Go.

To import individual metadata files in EpiCenter Manager, right-click a folder and select **Import** from the pop-up menu. This option is available only for metadata files that can be imported.

You can produce a new EpiCenter by importing all of the metadata files from an existing EpiCenter. To import all of the metadata:

1. Right-click the EpiCenter icon and select **Import Metadata** from the pop-up menu.
2. The default export file is **export.mdb** in your **ConfigFiles** directory. Click **Change** to select another file.
3. Select **Always Replace Existing Data** if you want the new data to replace existing entries of the same name.
4. Select **Continue after Errors** so that should an error occur, you can receive a partial report. To prevent the importation of excessive errors, the **Maximum [number of errors] Not Imported** is set to 100 by default.
5. Click **Go**.

Note the following:

- Re-importing objects does not delete references to those objects. For example, a measure definition can be re-imported without deletion of the measure mappings in Web pages that point to that measure.
- To speed up import and export, the entire Microsoft Access database is either read from or written to at once.
- All import operations are performed inside of a single database transaction. Any error that occurs during import can be rolled back completely.

See Appendix G, “Export/Import of Metadata” for more information.

TOGGLING DATAMART TABLES

To switch to the A set of tables from the B set, or vice versa, choose **EpiCenter\Toggle Datamart**, or click the **A or B** tables toolbar button. Queries run against the set of tables indicated by the enabled button, and extractions populate the set indicated by the disabled button.

The current datamart is shown at the bottom of the EpiCenter Manager window. See “Mirroring and History: A, B, X and Y Tables,” on page 96 for more information.

RUNNING THE SCRUTINY DEBUGGING TOOL

Scrutiny is an interactive debugging tool available within EpiCenter Manager. (It is also the command-line executable **scrutiny.exe**.) Scrutiny performs a superset of checks that the EpiPhany Application Server performs at startup time. Scrutiny provides output that you can examine instead of the AppServer (SRV) log file. If Scrutiny finds a problem, it attempts to fix it.

Scrutiny ensures that your EpiCenter (both EpiMeta and EpiMart) is in a consistent and functioning state. It does this by running an extensive set of queries against your EpiMeta and EpiMart to ensure various rules are followed and multiple descriptions of items are consistent. For example, Scrutiny can examine the EpiMart tables for missing indexes, missing rows, bad referential integrity, and so forth. It can also catch List Manager constraints such as not allowing group transaction filters for individuals. It also checks consistency constraints, such as the presence of **UNKNOWN** rows in EpiMart tables and well-formed references.

You can run Scrutiny periodically as a validation tool, or whenever you encounter any problems with metadata or EpiMart data, or if the Application Server fails to start. Scrutiny diagnoses and fixes many common (and less common) issues.

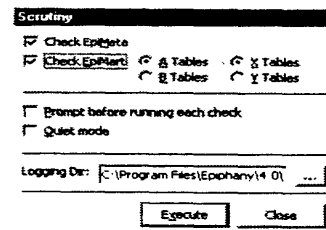
Choose **Run Scrutiny** from the **EpiCenter** menu. In the Scrutiny dialog box (Figure 100), select whether you want to perform EpiMeta or EpiMart checks, or both. Select the EpiMart tables you would like to check: A/B or X/Y. (By default, the value of the current datamart is selected.) You can select to have the program prompt you before issuing each type of check. In quiet mode, only errors and dots that mark Scrutiny's progress are displayed.

In the **Logging Dir** text box, the initial default logging directory is the location of EpiCenter Manager directory. You can enter a new directory in the text box, or click the button to browse for one. The EpiPhany system stores the last directory you have logged to in the Registry, and this becomes the new default.

Because many Web pages are incomplete for end use until options are set for their default report, Scrutiny checks for navigation nodes that do not have default reports. When Scrutiny runs, it displays warnings regarding any missing default reports. To remove these warnings, create a default report for every navigation node that has been configured.

In general, Scrutiny runs quickly (a few minutes for large EpiCenters), especially when EpiMart is not selected.

FIGURE 100: SCRUTINY DEBUGGING TOOL



When Scrutiny runs, it displays a text screen of the checks that are running. If an error is detected, it describes the nature of the error and either how to fix it yourself (usually in EpiCenter Manager) or proposes a solution for you. If you would like it to execute its proposed solution, press **y**. After Scrutiny has executed, it asks if you want to re-run that section, to ensure that the fix was applied successfully.

Warning: In some cases Scrutiny fixes are last resorts, and should be applied only if all else fails. These checks are identified as such, but be sure to read the descriptions carefully before applying any fix.

09625518.072500

09625518.072500

CONFIGURING WEB PAGES AND TOPICS

A user interacts with the E.piphany system by means of *Web pages*, which provide the user interface for your application. Web pages allow users to generate specific reports or perform specific actions. They are grouped into related sets of activities using *topics*. A *topic* consists of a set of Web pages and the links that you create between those pages.

Links within the E.piphany system perform the following functions, depending on how they are configured.

- They allow users to move from one Web page to another.
- They can carry report information (state) from one Web page to another. This state information includes the filters, attributes, options, and measures (FOAM) that a user has selected.
- They can automatically invoke the report-generation action of the destination Web page.

The combination of state information and report-invocation options associated with a link is referred to as the *behavior* of that link.

Topics are typically derived from templates called topic masters. Depending on the E.piphany solutions that you have purchased, you may have access to one or more topic masters.

The sections that follow discuss how to:

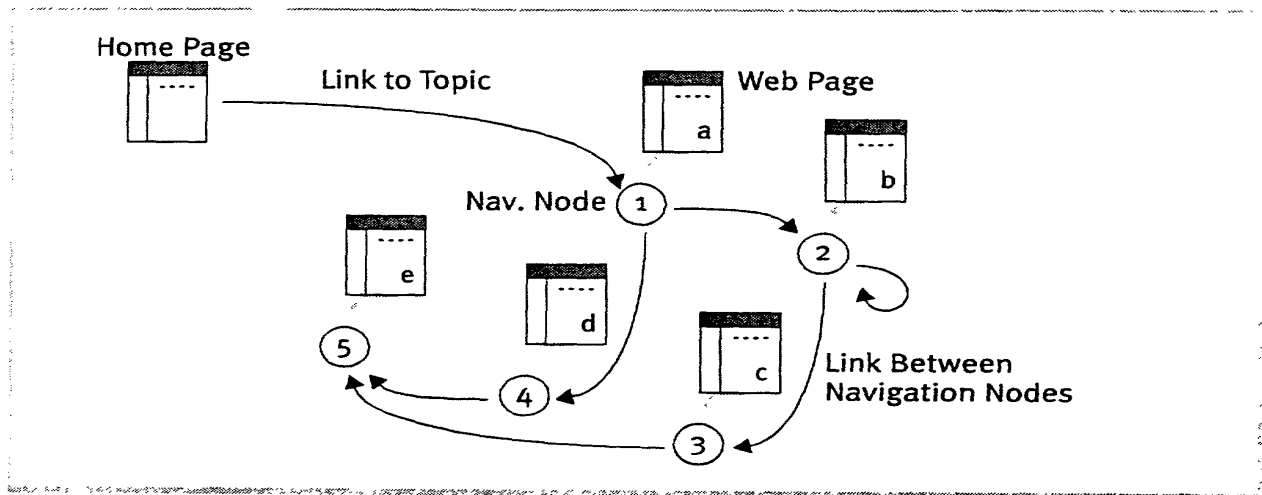
- Design a topic based on the user activities you want to provide
- Create the Web pages that perform each specific activity
- Build a topic from an existing topic master
- Assign Web pages to your topic
- Specify links between Web pages within a topic

- Add links to Web pages in other topics
- Provide site-specific on-line help for your topic and Web pages
- Create new topic masters

DESIGNING A TOPIC

The process of designing a topic is similar to planning an itinerary. You create a map of destinations called *navigation nodes*, the Web pages that are assigned to each node, and the routes or *links* between those nodes. Figure 101 shows the site map for a sample topic. A navigation node defines a location at which a user stops to perform an activity. The Web page that you assign to a navigation node determines the type of activity that the user can perform at that location.

FIGURE 101: SAMPLE TOPIC MAP



Next, you list the activities that you want your users to perform at each Web page, and the information that needs to be carried from one Web page to another along the way. Table 102, on page 305 shows a sample itinerary for a new topic.

FIGURE 102: SAMPLE ITINERARY FOR A NEW TOPIC

Nav. Node	Web Page	User Activity	Destination of Link	Linking Behavior at Destination
n/a	(Home Page)	Allow user to choose a topic	1	Clear state, display only
1	a	Report sales by region and quarter	2 4	Carry state, display only Carry state, execute report
2	b	Drill down (repeatedly) by product, salesperson, customer demographics, and so on	2 3	Carry state, execute report Carry state, execute report
3	c	Identify influential attributes	5	Carry state, display only
4	d	Create profiling charts	5	Carry state, display only
5	e	Show highest and lowest values	n/a	n/a

When you have mapped out a primary route, you can consider alternate routes, shortcuts, and side trips. The primary route typically includes the most comprehensive or complicated set of activities that you want your topic to support. The alternate routes support other related activities that the topic can support.

MAPPING USER ACTIVITIES TO WEB PAGES

The reports and data that your topic provides are determined by the Web pages that you configure. Each Web page has a specific type, which determines the reports, lists, or campaigns that users can create. The combination of filters, attributes, options, and measures (FOAM) that you configure for a particular Web page determines the range of data that your users can access through that page.

After you have configured a Web page, you can reuse it in other topics. If you need to create similar reports that give access to different data, you can create different Web pages of the same type. When you create more than one Web page of a given type, E.piphany suggests that you provide mnemonic names to those Web pages to indicate the scope of the data that each page can access.

Table 10 lists the Web page types that E.piphany provides for generating reports, lists, and campaigns. The Web pages that you configure for your users depend on the solutions that you have purchased as part of your E.piphany application.

TABLE 10: WEB PAGE TYPES FOR REPORTS, LISTS, AND CAMPAIGNS

(1 OF 2)

Web Page Type	Description
Advanced Rows and Columns	Display measures with respect to a pair of attributes and allow selection of multiple rows for drill-down reports
Basic Rows and Columns	Display measures with respect to a pair of attributes
Calendar	Display calendar of scheduled campaigns
Community Clusters	Identify groupings of values along a set of attributes
Cumulative Projections	Project cumulative results for a set time period based on previous time periods and current results so far
Group Campaign Segment to List Manager	Modify a group campaign segment with Group List Manager
Group Campaign to List Manager	Modify selection criteria for a group campaign with the Group List Manager

TABLE 10: WEB PAGE TYPES FOR REPORTS, LISTS, AND CAMPAIGNS

(2 of 2)

Web Page Type	Description
Group Campaigns	Create campaigns directed toward members of groups
Group List Manager	Create lists based on group demographics
High/Low Clusters	Find combinations of attributes associated with particularly high or low values.
Individual Campaign Segment to List Manager	Modify an individual campaign segment with Individual List Manager
Individual Campaign to List Manager	Modify selection criteria for an individual campaign with Individual List Manager
Individual Campaigns	Create campaigns directed toward individuals
Individual List Manager	Create lists based on individual demographics and transactions
Influences	Identify predictive relationships among a set of attributes
Lifecycles	Project the life cycle for a new product based on the performance of previous like products
Modeling	Identify predictive relationships for use in creating scored lists
Profiling	Chart values along an attribute, or compare values with different filters in effect
Schedule	Add a campaign to the calendar of scheduled campaigns
Scoring	Apply an ordering to a list to rank list members
Trends	Identify trends along a set of attributes and predict future values based on those trends
View List	View list members and download selected attributes
View List from Segment	View list members from within a campaign segment

Table 12 lists Web page types that are used to supply additional state information when users make a transition from certain Web page types to others. Web pages of the following types appear as intermediate nodes between an origination node and a destination node that displays a Web page that creates a report, list, or campaign.

TABLE 11: INTERMEDIATE WEB PAGE TYPES

Web Page Type	Description
Add a Filter Setting	This Web page type allows users to choose a set of filters to carry into the Web page that is next in the navigation path.
Add FOAM to ...	These Web page types allow users to choose specific filters, options, attributes or measures to carry into the next Web page of a given type.
Add Parameters to Trends	This Web page type allows users to choose parameters to carry into a Trends Web page.
Add Transaction Filters	This Web page type allows users to choose a set of transaction filters to carry into a List Manager or Campaign Manager Web page.
Select a Navigation Path ...	This Web page type displays navigation options without running a report. Path selector Web pages do not carry state. If you want to carry from one Web page to another, you must create a direct link from the source Web page to the destination Web page.
Top N Scores	Set a maximum size for a list to be scored by Scoring

Table 12 lists additional Web page types that perform other functions as part of your E.piphany application.

TABLE 12: OTHER WEB PAGE TYPES

Web Page Type	Description
Home Page	This is the home page for an E.piphany application.
Login Page	This is the login page by which users gain access to an E.piphany application. The login page is provided automatically. It does not appear in the list of Web page types in EpiCenter Manager.
Report Gallery	This is the repository for saved reports, lists, and campaigns.

CREATING WEB PAGES FOR USER ACTIVITIES

This section describes the process that you typically follow to create Web pages for the user activities that your topic is intended to support. Consult the next section, “Considerations for Specific Web Page Types,” on page 311, for information about creating specific Web pages.

1. In the **Presentation** folder of your EpiCenter, right-click the **Web Pages** icon, then choose **New Web Page**.
2. In the General tab of the Web Page dialog box:
 - a) Enter the name of your Web page.
 - b) Choose a Web page type from the **Web Page Type** drop-down menu
 - c) Enter a description of the Web page in the **Description** text box.

When you choose a Web page type, the contents of the dialog box adjust to allow you to enter information that is appropriate for that type.

3. If the **Attributes** tab appears, choose the attributes that you want to include in your Web page. Display-area buttons for the row- and column-attribute lists appear at the top of the **Attributes** pane. For example, in a Basic Rows & Columns Web page, the following display-area buttons appear: **Columns**, **Rows** and **Filters**.
 - a) Click on the name of a display area to select it. The attributes that are available appear in the **Object Gallery** pane, and you can drag items from the **Object Gallery** pane to the **Attributes** pane to add them to the display area you have selected.
 - b) If you have a large number of attributes for users to select from, you can use categories to reduce the size of the display area on the Web page. Click the **Category** button and choose **Add** from the drop-down menu. Enter the name of a new category, then drag attributes and drop them directly on the appropriate category name that appears in the **Attributes** pane. When the Web page is displayed, only the attributes that fall within the category that a user selects are shown. The **Category** button also allows you to remove or rename categories.
 - c) You can adjust the placement of attribute labels within the display area by clicking the **Sort**, **Up**, or **Down** buttons. You can adjust the indent level for an attribute label on the Web page by clicking the left-arrow or right-arrow button. To activate these buttons, choose an attribute in the **Attributes** pane.
4. If either the **Measure Layout** or the **Transaction Filters** tab appears, follow the same process that you used in the previous step to add measures or transaction filters to your Web page.

Note: EpiCenter Manager allows you to include measures in a Web page that do not necessarily join to attributes that appear on that page. To ensure that end users can obtain meaningful results from a Web page, be sure that the measures you select bear a useful relationship to the attributes that you have already selected.

5. If the Measure Sets tab appears, drag measures from the upper section of the **Object Gallery** to the **Measure Sets** pane. If you are configuring an Influences Web page, drag target attributes that you want to associate with a measure set from the lower section of the **Object Gallery** to the rows for the applicable measures.

When you have completed these steps, an icon for your new Web page appears in the Web pages folder in EpiCenter Manager.

If you have several Web pages of the same type to create, you can use an existing Web page as a starting template by following these steps:

1. Right-click the icon for the first Web page of that type and then choose **Duplicate** from the pop-up menu.
2. Enter the name of the new Web page in the **Duplicate: Web Page** dialog box.
3. Right-click the icon for the new Web page and choose **Edit** from the pop-up menu.
4. Delete attributes, filters, measures, and other settings that do not apply to the new Web page, and enter the information for those that do.

CONSIDERATIONS FOR SPECIFIC WEB PAGE TYPES

The sections that follow describe the basic functionality and configuration considerations for specific types of Web pages. If a Web page type is not listed, then the directions in the previous section should be sufficient for configuring a Web page of that type. For detailed information about each of the Web page types described in this chapter, refer to the on-line help pages in the following directory:

`C:\Program Files\Epiphany\instance\Web\help`

The help pages are in HTML format, and can be viewed with most Web browsers.

THE HOME PAGE

The Home page is configured automatically when you initialize your EpiCenter. It is updated automatically whenever you add a topic. The appearance of the home page for a particular user depends on the topics, favorite reports, and charts to which that user has access, as well as the User Preferences that she or he has set.

You can customize the home page as follows:

- By replacing the logo that appears in the upper-right-hand corner (applies to all Web pages, not just the home page)
- By adding a news banner that appears at the top of the page

Users can customize the page by specifying the number of favorite reports and charts to display and by saving favorite reports and charts. For more information about favorite reports and charts, refer to “Report Gallery,” on page 287.

REPLACING THE LOGO

The application logo appears in the upper-right-hand corner of every Web page (excluding pop-up-dialogs). To replace the logo with the logo for your company or solution:

1. Locate a GIF file that contains the logo you want to use.
2. Save a copy of the file called **clientlogo.gif** in the following directory:
`C:\Program Files\Epiphany\instance\WWWRoot\images`
3. Replace the **clientlogo.gif** file with the GIF file that contains the logo you want to use.

CUSTOMIZING THE NEWS BANNER

The Home page includes a banner that you can customize to display timely information to users. This information can include a message of the day, alerts about changes in the system, and the status of data loads. The news banner appears near the top of the page, above the list of topics and favorite reports.

To customize the news banner, edit the **homepage_news.template** file in the following directory.

```
C:\Program Files\Epiphany\instance\Web\Templates
```

By default, this file contains customization instructions, which E.piphany recommends that you replace.

The **homepage_news.template** file can include valid HTML, including applets. You can also include E.piphany AppServer macros (See Appendix A, "E.piphany Macros.") A particularly useful macro for the banner file is:

```
<!--EP CLASS="com.epiphany.presentation.DateMethod" language="en"
-->
```

which expands to become the date and time of the last successful extraction.

Each E.piphany solution includes a registered copy of the Category Ticker applet by Black Lab Design[®], which is an attractive and convenient tool for posting news.

Documentation for this applet is available in the **CategoryTickerDocs.html** file in following directory, and at <http://www.javapplets.com>.

```
C:\Program Files\Epiphany\instance\WWWRoot\help
```

The **homepage_news.template** file is parsed by the AppServer, not by the dynamic parser of the Web server, so the file must not include ASP scripts. You must use valid HTML syntax, which can include links to other dynamic pages. The contents of this file are included in the body of the Home page, so the file must *not* contain tags that belong outside of the body section of an HTML page. Avoid such tags as <HTML>, <HEAD>, <TITLE>, and <BODY>, along with the corresponding closing tags.

A sample **homepage_news.template** is included in the template directory. It uses the Category Ticker applet. If other applets are used, they should be placed in the **WWWRoot** directory.

THE LOGIN PAGE

The Login page is configured automatically when you initialize your EpiCenter. As with the Home page, you can include a banner that displays timely information.

To add a news banner, create or edit the **login_news.template** file in the following directory:

`C:\Program Files\Epiphany\instance\Web\Templates`

By default, this file contains customization instructions, which E.piphany recommends that you replace.

This file can include E.piphany AppServer macros, valid HTML body tags, and applets such as the Category Ticker applet, as described in the previous section.

Tip: E.piphany suggests that you provide instructions for logging in on the Login Web page. If a user must qualify her or his user name with the name of a particular domain in order to gain access to your E.piphany application, a note to this effect can be very helpful.

THE REPORT GALLERY

The Report Gallery is configured automatically when you initialize your EpiCenter. Refer to “Report Gallery,” on page 287 for details about managing saved reports with the Report Gallery. You can restrict the types of reports that appear in the Report Gallery for a given topic. Refer to “Configuring Navigation Nodes,” on page 337 for details.

INTERMEDIATE WEB PAGE TYPES

The intermediate Web page types listed in Table 11, on page 308 allow users to add state information to carry into a destination Web page. This information can include filters, options, attributes, or measures (FOAM), or other settings that affect the information that appears in a report.

These Web page types are available only in the Navigation Nodes dialog box. You do not configure new Web page contents for these Web page types. Instead, you choose the same Web page as the one that has been assigned to the navigation node into which the added state is to be carried. For instance, if you want to give users the option to add new filters when going from a Basic Rows and Columns or Advanced Rows and Columns Web page to an Influences Web page, you choose the same Influences Web page that is to receive that information as the Web content for the navigation node.

Intermediate Web page types are used between nodes when:

- the Web page at an origin node does not capture all of the information that is necessary to generate a report on the Web page that resides at the destination of a link.
- you want the report at the destination page to be generated automatically (the link to the destination node has the Create Report behavior).

For more information about when to use these Web page types, refer to “Configuring Intermediate Nodes,” on page 346.

SELECT A NAVIGATION PATH

The Select a Navigation Path Web page type derives its distinguishing properties from the navigation nodes to which it is assigned. You can create just one Web page for this type and reuse it throughout your E.piphany solution. Creating this Web page involves assigning a label, providing a description, and selecting the Web page type. This Web page type includes no filters, attributes, options, measurements, or other data elements.

BASIC AND ADVANCED ROWS AND COLUMNS

The Rows and Columns Web page types allow users to browse through the information that your datamart contains. Measures are broken out across a pair of attributes that the user selects. Values for one attribute appear as row headings in the report. Values for the other attribute appear as column headings. The measure values that correspond to the intersection of a pair of row and column attribute values appears in each table cell.

The Advanced Rows and Columns Web page type allows users to select multiple attributes to create a multiple-level, indented, drill-down report in a single step. The Basic Rows and Columns Web page type can produce the same report by adding attributes in successive drill-down reports.

It is possible for attributes and measurements that are unrelated to appear on the same Web page. E.piphany recommends that you take care to ensure that the attributes and measures you configure are relevant to each other. For details on configuring a Web page of this type, refer to “Creating Web Pages for User Activities,” on page 309.

COMMUNITY CLUSTERS, INFLUENCES, AND MODELING

The Community Clusters Web page allows users to identify clusters of records with similar characteristics. For example, you can use this Web page to find groups of customers who have similar demographic information. The Influences Web page allows users to identify attributes that have important impacts on a measure or another attribute. The Modeling Web page allows users to build predictive models based on predefined targets.

These Web pages use sophisticated decision-tree methods to analyze data. Depending on the size (cardinality) and number of attributes that users choose, the creation of these models can result in slow response times. You might need to consider limiting the number of attributes that you include in these Web pages.

When you configure any of these Web pages, you must identify the primary dimension by which to evaluate your data. For example, if your users want to analyze relationships or rankings among individuals, then the primary dimension is the individual. Similarly, groups might be another good candidate for a primary dimension.

To specify the primary dimension role, select the **Primary** button under the **Attributes** tab, then choose an attribute that encompasses an entire dimension, or a list-membership attribute, from the **Object Gallery** pane.

Next, you need to determine what attributes might be useful in analyzing this primary dimension. For instance, demographic attributes, such as age, income, or education level, may be useful attributes. The attributes that you include in must either come from the primary-dimension table or stand in a one-to-many relationship with that table.

***Warning:** When configuring a Community Clusters, Influences, or Modeling Web page that includes source attributes which are not derived from the primary dimension, you must ensure that the one-to-many relationship between source attributes and the primary dimension holds. EpiCenter Manager does not enforce the requirement of a one-to-many relationship, but if this type of relationship does not exist, then some or all queries attempted by users from these Web pages can fail.*

To add attributes, click the **Source Attributes** button and then drag and drop attributes from the **Object Gallery** pane. Restrictions on the types of attributes that you can include appear at the bottom of the **Object Gallery** pane.

Finally, you need to specify a Measure Set to be used.

For a Community Clusters Web page, choose a Measure Set of type **Clustering** that contains a single **COUNT** measure that counts the number of distinct entities (e.g., individuals) in the primary dimension table. For example, if the primary dimension is individuals, then the measure set must contain a single **COUNT** measure counts of the number of individuals.

For an Influences or Modeling Web page, the Measure Set can be of type Regression or Classification. When you use a Regression Measure Set, you must specify a COUNT and a SUM role. You can also specify an optional SUMSQUARED role.

If you use a Classification measure, you must also specify a target attribute to use in conjunction with that measure. The target attribute is the attribute that the Influences or Modeling Web page attempts to predict. To select a target attribute, drag it from the **Object Gallery** pane and drop it in the **Target Attribute** folder that appears directly to the right of the Classification measure set that you selected earlier.

If you would like to analyze relationships that are not contained in the primary dimension, such as clusters of individuals based on whether or not they have bought products X, Y, or Z, you can work around the restriction that attributes can come only from the primary dimension.

To do so, you can create supplemental attributes in the individual table that are populated at extraction time and indicate which individuals bought these products. Note however, that these attributes are not automatically synchronized with respect to other dimension values. As dimension values are updated, these attributes can be rendered inconsistent unless you take steps to refresh them from time to time.

Community Clusters, Influences, and Modeling Web pages perform complex statistical analyses that can take a long time to run. The actual running time depends on the number of attributes selected by the user on the Web page, the number of values that these attributes have, and the number of rows in the primary dimension.

As a result, when configuring Web pages of either type, E.piphany suggests that you omit unneeded attributes that have a high cardinality, since these attributes make the application take longer to run and generally have little impact on the results of a report.

USING LISTS AS ATTRIBUTES IN COMMUNITY CLUSTERS, INFLUENCES, OR MODELING WEB PAGES

When the primary dimension of a Community Clusters, Influences, or Modeling Web page is the same as the individual or group dimension, your Web page can use list membership as an attribute that indicates the group in which an individual is a member.

SLOWLY CHANGING DIMENSIONS

When the primary dimension of a Community Clusters, Influences, or Modeling Web page uses the Slowly Changing Dimensions semantic type, a single logical member of the dimension (such as a single individual) may appear as multiple rows in the dimension table if the attribute values of that dimension member have changed over time (for example, an individual got married or moved to a different state).

In this case, Influences and Modeling treat each row in the dimension table as a different member of the dimension when building a model. This may occasionally lead to minor differences in counts between these Web pages and the List Manager Web pages, but should not present any major difficulties.

If you are configuring a Web page that includes measures that perform COUNT DISTINCT operations on fact tables, the dimensions that query those measures can report duplicate values when they contain slowly changing dimensions. To avoid this problem, you can add a column to each of those dimensions that contains a duplicate of the *sskey* value and configure measures that perform COUNT DISTINCT operations on those columns.

AGGREGATES

If you build an aggregate on the primary dimension of a Community Clusters, Influences, or Modeling Web page that includes all columns of the base dimension (or at least all columns that are used as attributes or filters on the Web page), queries started by that Web page almost always use the aggregate.

PERFORMANCE ISSUES

Two kinds of queries cause Community Clusters, Influences, and Modeling Web pages to adopt complicated query plans: queries involving membership in lists as source attributes or targets, and regression queries in Influences and Modeling that are filtered on dimensions other than the primary dimension. All other things being equal, these kinds of queries tend to take somewhat longer than ordinary queries. If the processing time for queries is a concern to your installation, you may want to avoid these two types of queries.

CUMULATIVE PROJECTIONS

Cumulative Projections Web pages allow users to project results for the current time period (typically a quarter), based on results that have been recorded so far within the period and the pattern of results from previous periods. In Cumulative Projections Web pages, you must assign only relative-date attributes to the Rows display area, such as Days until End of Fiscal Quarter, or Weeks until End of Fiscal Quarter, or Days until End of Month.

GROUP OR INDIVIDUAL CAMPAIGN MANAGER, CALENDAR, AND SCHEDULE

Campaign Manager Web pages allow users to create campaigns based on individual or group attributes. The prerequisites that apply to Group List Manager and Individual List manager also apply to Campaign Manager Web pages. Refer to “Prerequisites for Creating List-Manager Web Pages,” on page 325 for information about these prerequisites.

Epiphany recommends that you configure at least one list-management Web page before you configure a Campaign Manager Web page. See “List Manager Web Pages,” on page 323 for details.

In addition to the prerequisites for list-management Web pages, Campaign Manager Web pages have the following requirements:

- The **Campaign** and **Cell** base dimensions must be available.
- The **campaign** and **cell** dimension roles must be available.
- You must create attributes for each of the columns in the **campaign** and **cell** dimension roles.

These base dimensions and dimension roles are created automatically when you initialize an EpiCenter with **Include List Manager** checked in the Initialize EpiCenter dialog box.

To create a Campaign Manager Web page:

1. Open the Web Page dialog box by right-clicking the Web Pages icon and choosing New Web Page from the pop-up menu.
2. Fill in the General tab. Choose Group Campaigns or Individual Campaigns as the Web page type.
3. In the Attributes tab, click the **Campaign** button to select the **Campaign** display area. Drag all attributes in the **campaign** and **cell** dimension roles into the **Attributes** list box.
4. Ignore the **Preview** button. Attributes in the **Preview** display area are ignored by Campaign Manager Web pages.
5. Configure filters on attributes of the **group** and **indiv** dimension roles by clicking the **Filter** button and dragging and dropping filters from the Object Gallery. Filters on these attributes are termed demographic filters. Filters on other dimension-role columns are suppressed at run time by the AppServer.
6. Ignore the Transaction Filters tab. Transaction filters are used only in list-management Web pages.

You can customize the labels that the Web page displays for the various factors that are used to calculate the costs and expected returns for a campaign. To do so:

1. Open the **Solutions** folder, then the **Options** folder, and then the **SegmentationCalcs** option.
2. Enter a glossary entry and description for this option in the **General** tab, then open the **Values** tab.
3. In the **Values** tab, choose a value from the **Values** list box and then enter a new label for that value in the **Label** text box.
4. Repeat the previous step for each value in the list.

HIGH/LOW CLUSTERS

High/Low Clusters Web pages allow users to identify the highest and lowest values for a set of attributes with respect to a measure. These Web pages display only one list of attributes, rather than the pair of attribute-lists that appear in the Rows and Columns Web pages. The **Attributes** tab for a High/Low Clusters Web page includes the **DimSingle** display-area button and **Filters** display-area button.

LIFECYCLES

Life Cycles Web pages allow users to project future values based on patterns established by previous long-term cycles. For instance, if you have sales records for previous products, your users can use the performance figures for those products over time as a model for predicting the performance of a new product that they plan to introduce.

You must assign only absolute date attributes to the Columns display area. You can assign attributes of any type to the Rows display area.

Note: The columns in Lifecycle Web pages should be absolute date columns, such as the fiscal year or the month of a particular year, such as January 1998. They should not be relative or cyclical columns such as month (that is, January, February, and so forth).

LIST MANAGER WEB PAGES

The following Web page types allow users to derive lists of groups or individuals, such as customers, resellers, households, contacts, or other similar entities:

- Group List Manager
- Group Campaign to List Manager
- Group Campaign Segment to List Manager
- Individual List Manager
- Individual Campaign to List Manager
- Individual Campaign Segment to List Manager

Users can generate lists based on demographic data and transaction histories. Demographic data includes attribute data that appears in your Group or Individual attribute tables. Transaction histories are derived from fact tables.

Unlike most data-warehousing applications, list-management applications use fact data (transactions) to classify information about customers, prospects, and so on. (In most other cases, attribute data is used to classify facts, such as by breaking out sales data by customer.) List members can be selected or excluded based on their participation in a particular type of transaction.

The list-manager Web Page types provide sophisticated list-management capabilities, such as:

- Demographic filtering based on attributes of the **group** or **indiv** dimension roles
- Transaction filtering based on measures and attributes of other dimension roles
- Generation of approximate and exact member counts

The **group** and **indiv** dimension roles are provided for you when you configure your EpiCenter for lists and campaigns (if you ensure that the **Include List Manager** check box of the Initialize EpiCenter dialog box is checked). These dimension roles provide attributes that users can employ to break out lists for various purposes.

DEMOGRAPHIC FILTERS

A user of a list-manager Web page applies filters to restrict list membership based on attributes of the **group** or **indiv** dimension roles. Filters based on these attributes are called *demographic filters*. After the user selects a filter in a List Manager Web page, he or she can find out how many matches it corresponds to in the database. These filters are configured in the same manner as filters in other Web pages.

TRANSACTION FILTERS

You can configure list-manager Web pages to allow users to filter individual and group dimensions based on participation in transactions. A *transaction* is an action that an individual or a member of a group might participate in, such a purchase, a return, a call to a call center, the receipt of a treatment, or a response to a campaign. Users can query against measures along these types of transactions. For instance, they might want to know how much a customer has purchased, how many campaigns a prospect has received, or how often the members of a household place calls to a call center. They might also want to know which customers have *not* participated in a certain type of transaction.

To allow users to filter on transaction data in this manner, you need to define *transaction filters* in addition to regular filters. Refer to “Configuring List-Manager Web Pages,” on page 328 for information about configuring transaction filters.

PREREQUISITES FOR CREATING LIST-MANAGER WEB PAGES

If you want to provide EpiPhany list-management and campaign-management capabilities to your users, you must be sure that your EpiCenter has been configured with the built-in fact tables, dimension tables, and dimension roles that support these capabilities. These structures are only built when the **Include List Manager** check box is checked in the Initialize EpiCenter dialog box at the time you initialize your EpiCenter. To verify that these supporting structures are in place:

1. Open the **Schema** folder and then the **Base Dimensions** folder, then look for the **GroupDim** and **IndiviDim** base dimensions.
2. Open the **Dimension Roles** folder and look for the **group** and **indiv** dimension roles.
3. Open the **Facts** folder and look for the **Communication** and **Ind_Group_Joiner** fact tables.

If these tables are not present, you must initialize a new EpiCenter. In order to preserve your work so far, export your metadata. Refer to Appendix G, “Export/Import of Metadata,” for information on how to import this metadata into a new EpiCenter.

Make sure that you have defined extraction jobs to populate these tables.

ENABLING APPROXIMATE COUNTS

The list-manager Web pages provide a quick-count feature that uses sampling to estimate the counts for subsets of lists that fit the selection criteria a user has chosen. These estimates allow the queries that calculate these counts to complete in the time it takes to evaluate the sample as opposed to the entire dimension.

You can enable this feature by assigning a positive value to the **min_sample_invlog10** option in the General tab of the Configuration dialog box (see “General Settings,” on page 417). The value that you set in this option corresponds to the inverse log (base 10) of the sampling probability. For example, a value of 2 would create a 1-percent sample ($1/10^2$).

THE IND_GROUP_JOINER FACT TABLE

The *Ind_Group_Joiner* table is a special-purpose fact table that defines the many-to-one relationship between individuals and groups. This table identifies demographic dimensions and assigns individuals to groups. Each of these actions is discussed below.

- Identifying demographic dimensions

The Momentum Builder program and the Application Server use the **indiv** and **group** dimension roles associated with the **Ind_Group_Joiner** table to identify the individual and group base dimension tables.

- Assigning of individuals to groups

The data in this fact table determines which individual belongs to which group.

005220.072500

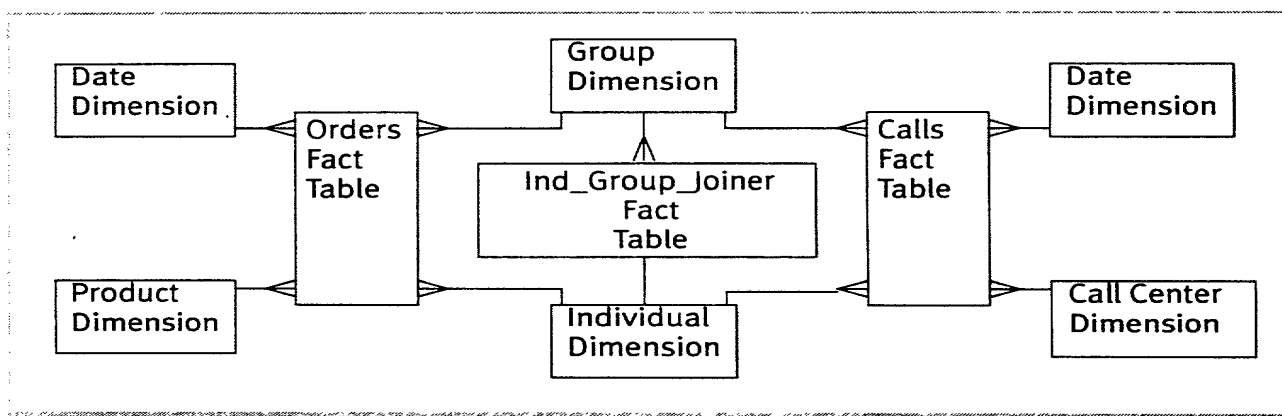
- Defining who appears in lists and campaigns

The table also defines the individuals and groups that are a part of the selection pool for lists and campaigns. Any individual or group that does not have its key in this table is not available for inclusion in a list or campaign. Even if you have individuals that belong to no group or groups without individuals, you still need to enter their keys in this table to include them in your lists and campaigns.

Table 103, on page 327 illustrates a typical star schema for E.piphany list-generation data. In this example, there are two regular fact tables called **Orders** and **Calls**. There are also dimension tables associated with each of those facts in addition to the **Group** and **Individual** dimension tables. The **Ind_Group_Joiner** table is a third fact table that implements the many-to-one relationship between the **Individual** and **Group** dimensions. With a schema such as the one shown, you could generate lists based on:

- Demographic data in either the group or Individual dimension tables
- The total amount of orders that have been placed by an individual
- The combined total of orders per group (such as a household or company)
- The total number of calls in a given time period for an individual or group placed to a given call center

FIGURE 103: SCHEMA FOR LIST-GENERATION DATA



FACT-TABLE CLUSTERS AND COUNTS

Clusters and counts make transaction filtering faster. A *cluster* is a copy of the fact table sorted on an attribute. A large table should be sorted on disk with its leading term being the most selective filter.

Counts keep statistics about how often entries for a group or individual appear in a fact table. Counts are used by the query engine to select the best-clustered copy of a fact table.

To enhance the performance of queries that use transaction filters, choose at least one cluster for each fact table that contains transaction data that your users filter against. Be aware that clusters consume the same amount of disk space as the fact tables from which they are copied. You must define a cluster in order to define counts for a fact table.

CONFIGURING LIST-MANAGER WEB PAGES

After you enter a name, choose a Web page type, and enter a description, the actions involved in configuring a list-manager Web page include:

- Adding attributes to the Preview display area of the Attributes tab
- Adding demographic filters in the Attributes tab
- Adding transaction filters in the Transaction Filters tab

The sections that follow discuss these actions.

CONFIGURING ATTRIBUTES

You set up attributes for list-management Web pages using the Attributes tab. Restrictions on the types of attributes that you can include appear below the Object Gallery list box. Although the **Campaign** button appears, campaign attributes are unused in list-management Web pages. Preview attributes are required. You configure these attributes by clicking the **Preview** display-area button and then dragging and dropping attributes from the **Object Gallery** pane.

If the Web page type is Group List Manager, Group Campaign to List Manager, or Group Campaign Segment to List Manager, then only group attributes are allowed. If the Web page type is Individual List Manager, Individual Campaign to List Manager, or Individual Campaign Segment to List Manager, the attributes can pertain either to an individual or the group of which that individual is a member.

CONFIGURING DEMOGRAPHIC FILTERS

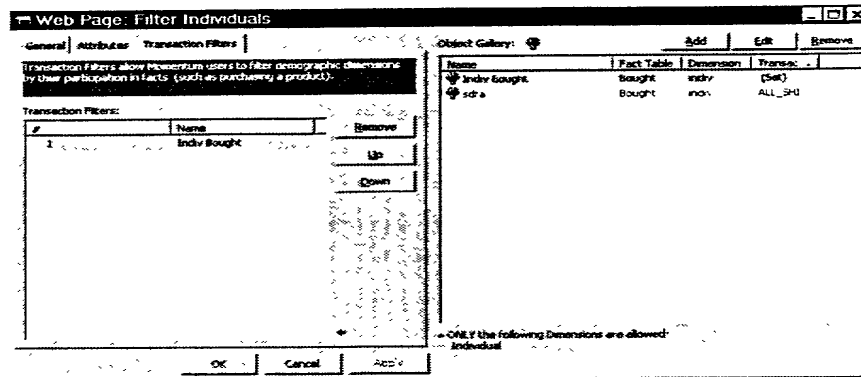
Demographic filters are based on attributes of the **group** and **indiv** dimension roles. In other respects, they are just like the filters used in other Web pages. If you include nondemographic filters in the Filters display area, those filters are suppressed by the AppServer at run time. You can include filters for other types of dimensions when you configure transaction filters.

CONFIGURING TRANSACTION FILTERS

Take the following steps to configure transaction filters for a list-manager Web page:

1. Open the Transaction Filters tab of the Web Page dialog box for a Web page of type Filter Groups or Filter Individuals. Figure 104, on page 330 illustrates this tab. Click the **Add** button to add a new transaction filter. Click **Edit** or double-click a row in the Object Gallery to edit an existing transaction filter.

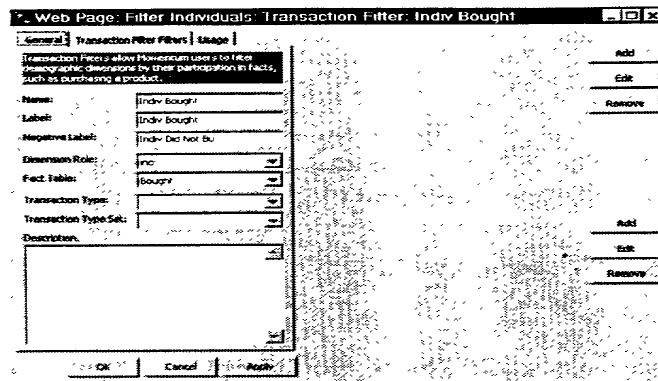
FIGURE 104: TRANSACTION FILTERS TAB OF THE WEB PAGE DIALOG BOX



2. In the General tab of the Transaction Filter dialog box (Figure 105), enter the following information about the transaction filter:
 - The name

When you enter the name, EpiCenter Manager fills in values for the label and negative label. The negative label is the name of a filter that excludes values that qualify for the transaction filter.
 - The dimension role (either **group** or **indiv**)
 - The transaction type to apply to the transaction filter
 - The transaction type set
 - The description for this transaction filter

FIGURE 105: TRANSACTION FILTER DIALOG BOX



3. In the Transaction Filter Filters tab (Figure 106, on page 332), drag and drop the attributes and measures that you want to allow in this transaction filter. To add new candidate attributes, click **Add** near the Attributes text box. To add new measures, click **Add** near the Measures text box. To edit an existing attribute or measure, choose it and then click the appropriate Edit button. You can include measures in your transaction filters.

[illegible]

- Opportunity
Grants
Engineering
Graduate
Incentive

Re

2

- 10

PROFILING

The Profiling Web page allows users to create charts and perform comparisons by applying combinations of attributes and filters. Profiling Web pages use only one display area for attributes. These Web pages are not adversely affected by high-cardinality attributes. Users can use a variety of filters to create comparison reports, so EpiPhany suggests that you include other filters that might be of use.

SCORING

The Scoring Web page allows users to rank members of a list of groups or individual using a predictive model that they have created with either a Community Clusters, Influences or Modeling Web page. Users also can score lists based on a measure. You only need to create one Scoring web page in your EpiCenter. Refer to “Configuring Navigation Nodes for Scoring Web Pages,” on page 349 for information on how the Scoring Web page is used in topics.

TRENDS

The Trends Web page allows users to calculate projections based on trends that are inherent in the data with respect to a set of attributes. The Trends Web page has two display areas, **Columns** and **Rows**. You must assign only absolute time attributes to **Columns**. You can assign any attribute to **Rows**.

VIEW LIST AND VIEW LIST FROM SEGMENT

The View List and View List from Segment Web pages allows users to display and download selected attributes for list members. This page also provides access to other list-related Web pages for scoring, updating selection criteria, and creating campaigns.

TOPICS AND TOPIC MASTERS

A topic master is the template for a topic. E.piphany provides topic masters that embody the solution maps for on-line analytical processing (OLAP), database marketing, and e-commerce applications. The topic masters that you have available depend on the E.piphany solutions that you have purchased.

A topic master specifies a mapping of navigation nodes and links. A navigation node is a location within the topic at which a Web page can reside. Each navigation node in a topic master has an associated Web page type. When you create a topic from a topic master, you assign a Web page of the appropriate type to each navigation node that you choose to include in your topic.

You do not need to assign a Web page to a navigation node if your topic does not require a Web page of that type. You can either remove the navigation node from the topic or leave it unassigned. The AppServer displays links to navigation nodes that do not have Web pages assigned to them, but it disables those links. However, the AppServer does *not* display links to nodes that users do not have permission to visit. You can hide disabled links either by omitting the target nodes of those links from groups to which users have access, or by unchecking the Visible check box associated with each disabled link in the Links tab of the Navigation Node dialog box. Refer to “Assigning Links and Selecting Behaviors,” on page 341 for details.

If a navigation node is inaccessible, links from that node to other nodes are disabled. If all of the navigation nodes with links to a node that your topic needs are disabled, users cannot reach that node. In a situation like this, you can add a link to the node that you require from another node that is still reachable by users.

BUILDING A TOPIC

The process of building a topic includes the following stages:

1. Creating Web pages for the navigation nodes that you intend to use (if you have not done so already)
2. Creating the topic from a topic master
3. Modifying the structure of the topic by adding or removing navigation nodes (optional)
4. Assigning Web pages to navigation nodes (for nodes that do not have Web pages assigned by default)
5. Choosing behaviors for the links between Web pages

You can also create topics from scratch, without using a topic master as a template. However, E.piphany does not recommend that method. Instead, it is often more convenient to use the topic masters that E.piphany has supplied as part of your application.

CREATING A TOPIC FROM A TOPIC MASTER

To create a new topic, take the following steps:

1. Expand the **Presentation** folder for your EpiCenter in EpiCenter Manager, right-click the **Topics** icon, and then choose **New Topic** from the pop-up menu to bring up the Topic dialog box, as shown in Figure 107, on page 336.
2. In the **General** tab, add the following information:
 - a) Enter the name of your new topic in the **Topic Name** text box. As you do, EpiCenter Manager fills in the **Singular** and **Plural Label** text boxes. Edit the contents of these boxes as needed.
 - b) Choose a topic master from the **Topic Master** drop-down menu.
 - c) Pick a color palette for the topic bar from the **Palette** drop-down menu. Then pick a color from the **Color** menu.

- d) The default background image is suitable for most applications, and E.piphany suggests that you leave the **Background Image** text box set to the default value. Enter a description of the topic including the intended audience and usage.
- e) Leave the **Visible** check box checked, unless you do not want this topic to be accessible from the home page.
- f) Click the **Apply** button to initialize your topic.

FIGURE 107: THE TOPIC DIALOG BOX

Topic: Enterprise Database Marketing (5/10)

General | Navigation Nodes | Topic Order

A topic is a collection of navigation nodes.

Topic Name: Enterprise Database Marketing (5/10) ☒ Visible

Topic Master: Enterprise Database Marketing

Singular Label: Enterprise Database Marketing (5/10)

Plural Label: Enterprise Database Marketing (5/10)

Palette: Cool Palette

Colors: Slate

Background Image: background.gif

Description:

3. In the Topic Order tab, use the **Up** and **Down** buttons to adjust the placement of the topic bar on the Home page.
4. Click **Apply** to apply the settings you have selected to your topic.

At this point, you have created a topic with a default set of navigation nodes and links.

The **Update** button allows you to refresh the topic from the topic master from which it has been created. When you refresh a topic, some of updates that you have made to that topic are lost.

CONFIGURING NAVIGATION NODES

This section discusses the process for configuring the navigation nodes in a topic, which involves the following activities:

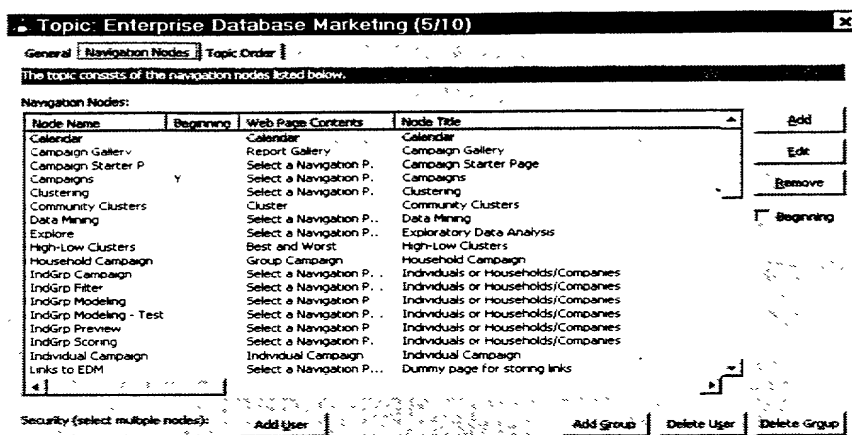
- Assigning Web pages to navigation nodes
- Configuring links and link behaviors
- Providing access to users and groups
- Optionally configuring additional navigation nodes for a topic
- Optionally hiding or deleting unneeded navigation nodes
- Providing on-line help for your topics and Web pages

If you have created only one Web page of a given Web page type, EpiCenter Manager assigns that Web page to each node in your topic that is configured to display a Web page of that type. Default links and link behaviors are inherited from the topic master that you choose. These default settings allow you to complete the process of configuring navigation nodes by making selective adjustments, rather than having to manually indicate every node and link.

The Navigation Nodes tab displays the name, the Web page to be displayed, and the title of each navigation node in a list box. The **Node Title** column shows the text that appears at the top of the Web page that is displayed at a node (Figure 108, on page 338). To reach this tab if it is not currently open, right-click the topic icon in the Presentation folder of your EpiCenter, then choose **Edit** from the pop-up menu, and then click **Navigation Nodes**.

Configuring a navigation node involves assigning Web page contents, granting access to users and groups, choosing Report-Gallery properties, and configuring links. The remainder of this section describes all of these actions except the configuration of links, which is discussed in the next section.

FIGURE 108: THE NAVIGATION NODES PAGE OF THE TOPIC DIALOG BOX



Each topic must have a single beginning node. This node is displayed when a user follows the link to a topic on the Home page. A **Y** in the **Beginning** column of the **Navigation Nodes** list box indicates the beginning node for the topic.

Although you can choose a beginning node by selecting a node from the list box and clicking the **Beginning** check box, E.piphany recommends that you use the beginning node that has been set by default in the topic master.

ASSIGNING WEB PAGES TO NAVIGATION NODES

The **Web Page Contents** field of the **Navigation Nodes** tab displays the name of the Web page that is assigned to a navigation node. If this field is blank, you must either:

- Create a Web page of the appropriate type and then assign it to the node
- Choose from among the several Web pages of that type that are currently available and assign it to the node
- Delete the node if it is unneeded, or assign a different Web page type to that node

To assign a Web page to a navigation node, take the following steps:

1. Double-click the row that displays the node to which you want to assign a Web page, or select that row and click **Edit**. Either action opens the Navigation Node dialog box shown in Figure 109.

FIGURE 109: THE NAVIGATION NODE DIALOG BOX

Topic: Enterprise Database Marketing (5/10): Navigatio...

General | Links | Access | Report Types | Reports | Usage

A navigation node is a single form for end user interaction.

Node Name: Calendar

Node Title: Calendar

Web Page Type: Calendar

Web Page Contents: Calendar New...

Prompts:

Description:

OK Cancel

2. If a Web page that produces the specific report you require does not already exist, click **New** and then follow the instructions in “Creating Web Pages for User Activities,” on page 309 to create it.
3. You can change the Web page type that is assigned to a navigation node by choosing a type from the **Web Page Type** list box. Table 10, on page 306 contains a list of Web page types that generate reports. Table 12, on page 309 contains a list of Web page types that perform other functions. For ease of maintenance, E.piphany suggests that you use discretion in changing the Web page type assigned to a navigation node.
4. Choose a web page from the **Web Page Contents** list box. This list box displays only Web pages that can be used as the content of the chosen Web page type.
5. Enter a prompt that describes the actions that you want users to perform at this node. This prompt appears near the top of the Web page.

6. Enter a description of the node in the **Description** text box.
7. The **Access** tab displays the groups and users who currently have access to the navigation node. Use the **Add Group** and **Add User** buttons to give access to selected groups and users. To remove access, choose a group or user from the appropriate list box and click the corresponding **Remove** button.
8. The **Report Types** tab displays the types of report files that the Report Gallery displays when accessed from the current navigation node. Use the **Add Types** button to add report types. Select a report type from the list box and click **Remove** to eliminate it from the display for this node. To restrict reports that the Report Gallery displays to those in the current topic, check **Only show reports from this topic**.
9. The **Reports** tab lists all reports that have been saved from the current navigation node. The **Remove** buttons allow you to remove a selected report. The **Edit** button allows you to update the properties of a selected report.

CREATING DEFAULT REPORTS

E.piphany recommends that you create a default report for the Web page you configure at each navigation node. You can create a default report for a Web page by:

1. Starting the Application Server. Do this after you have completed configuring your application and run Scrutiny to validate your configuration.
2. Navigating to the node. Your Web browser displays the Web page at that node.
3. Choosing the filters, options, attributes, and measures that shall act as default settings.
4. Clicking the **Save** button to display the Report Gallery

5. Navigating to the **Public/Defaults** folder and saving the report there. The name that you choose does not matter, although a mnemonic name makes it easier to distinguish the default report for one node from another node of the same Web page type.

Note: When you create a default report for a node that displays a High/Low Clusters Web page, be sure to open the compare to other filters / measurement pop-up and fill in a default for the comparison measure. E.piphany suggests that you use the same measurement as the default measurement that appears on the main Web page. Set this comparison measurement even if the default report uses the disproportionately or absolutely comparison method, as a user might select compare to other filters/measurements but forget to fill in a comparison measure.

ASSIGNING LINKS AND SELECTING BEHAVIORS

Links provide the means for users to jump from the Web page at one navigation node to the Web page at another node. Each navigation node has a set of links that have been assigned to it by default from the topic master. The Links tab of the Navigation Node dialog box allows you to assign links to navigation nodes. These links specify the destination nodes and behaviors to perform as part of the transition from the current node to a destination node.

Links are assigned to the navigation nodes from which they originate. The navigation nodes that are derived from a topic master include a number of links that are provided by default. In general, you only need to assign links to nodes or adjust the behavior of a link when you:

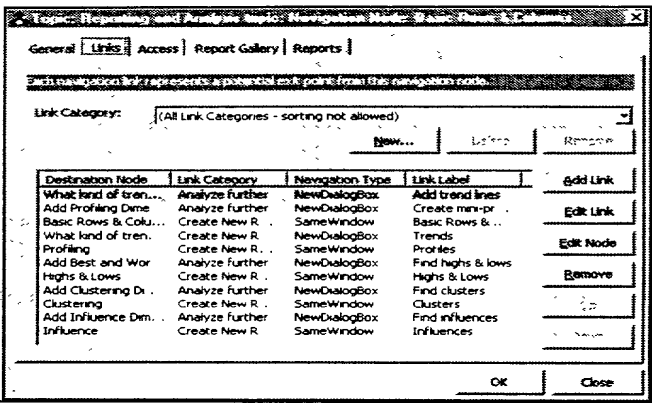
- Add new nodes
- Delete existing nodes
- Provide new, alternative links that appear in addition to the links that are derived from the topic master

If you choose, you can display alternative links in a separate list on the Web page by assigning them to a new link category.

You assign new links, adjust the behavior or destination of existing links, add links to existing categories, or create new categories of links with the Links tab of the Navigation Node dialog box. To reach this tab if it is not already open:

1. Open your topic in the **Topics** folder, which resides in the **Presentation** folder for your EpiCenter in EpiCenter Manager.
2. Open the **Navigation Nodes** tab.
3. Double-click the row for a node in the **Navigation Nodes** list box, or select a row and click **Edit**, to display the **Navigation Node** dialog box.
4. Click the **Links** tab. Figure 110 shows the contents of this tab.

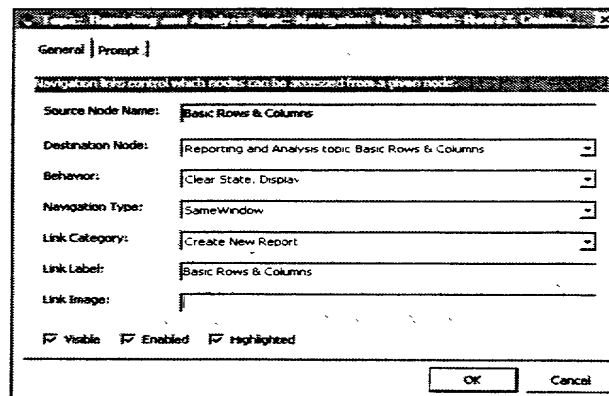
FIGURE 110: THE LINKS TAB OF THE NAVIGATION NODE DIALOG BOX



By default, all of the links that have been assigned to a node appear in the list box. You can view the links for a specific link category only by choosing a category from the **Link Category** list box. When you view the links in a specific category, you can rearrange the order in which they appear using the **Up** and **Down** buttons. Links in a category are displayed to the user in the order in which they appear within this dialog box.

You can create a new link category by clicking **New**. You can rename or delete a link category by selecting it from the list box and clicking **Rename** or **Delete**. To add a new link, click **Add Link**. To edit an existing link, select the link and click **Edit Link**. When you add or edit a link, the **Navigation Link** dialog box (Figure 111, on page 343) appears.

FIGURE 111: THE NAVIGATION LINK DIALOG BOX



The name of the current node is displayed under **Source Node Name** in the **Navigation Link** dialog box. It cannot be edited. To set the properties of the link:

1. Choose a destination from the **Destination Node** list box. The link leads to the node that you indicate. The destination node can be in any topic, and the list box lists both the topic and node names of all available nodes.
2. Choose a behavior from the **Behavior** list box. Each behavior includes a pair of components. The first component indicates whether to carry the state information that a user has selected in the current Web page to the Web page at the destination node. The second component indicates whether or not to generate a report when opening the destination Web page.

The behavior components describe the actions that are performed when the link is followed. Table 13 describes the behavior components. The combinations of components that are available for a particular link depend on the type of Web page that resides at the destination node.

TABLE 13: LINK BEHAVIOR COMPONENTS

Behavior Component	Description
Carry State	This behavior component specifies that the state of the current web page (which includes such things as options, filter settings, selected attributes, and lists) should be preserved for use by the destination Web page
Clear State	indicates that the state should not be carried to the destination Web page
Create Report	indicates that a report should be generated (generally by means of a datamart query) for display by the web page at the next node.
Display	indicates that the web page at the next node should simply display information without generating a database query.

3. Choose a navigation type. This indicates the type of window in which the next web page will be displayed. Table 14 describes the available options.

TABLE 14: NAVIGATION TYPES

Navigation Type	Description
Same Window	The destination node is displayed in the same window as the current node.
New Browser Window	The destination node is displayed in a new browser window. In this case, the current node remains in the current browser window.
New Dialog Box	The destination node is displayed in a new pop-up window. The current node remains in the current browser window.
Opener Window	The destination node is displayed in the window from which the current node was opened. For example, if the current node is a popup window, this displays the destination node in the window from which the popup was opened.

4. Choose a link category from the **Link Category** list box or create a new one by clicking the **New** button. Link categories allow you to group links that you want to appear together on the Web page. Links are typically displayed in the left-hand side bar as a tabular list, with the category name shown as the list heading.

Some link categories are treated as special cases by certain Web pages. These special-purpose categories are described in Table 15. If a Web page does not recognize a link category as a special case, it displays the links in that category in the side bar, in the same fashion as any other link category.

TABLE 15: SPECIAL-PURPOSE LINK CATEGORIES

Link Category	Web Pages	Placement and Usage
Analyze Further	Basic Rows and Columns, Cluster, Influences, Lifecycles, Profiling, Quarter Projections, Scoring, and Trends	Links appear in the results section of the Web page only.
Create New Report	All	Links appear in the report section of the Web page only.
Multiple Choices	Select a Navigation Path	Links appear in the body of the Web page.
Wizard Steps	All	Links appear at the bottom of the report section.
You Are Here	All	Links appear in the report section of the Web page only.

5. Enter a label in the **Link Label** text box. This text appears as the link.
6. Check the **Visible** check box if you want to have the link visible to the user.
7. Check the **Enabled** check box to enable the link. If **Visible** is checked and **Enabled** is unchecked, then the user will see the link label but will not be able to follow the link.
8. Check **Highlighted** to display the link with the background color of the topic.

- ## CONFIGURING INTERMEDIATE NODES

To ensure that users get the results they desire, you can configure an intermediate node with a Web page of the appropriate Add FOAM type for your destination. Users can use the intermediate page to enter the information that they want to pass through to the destination page. When you create the navigation node for an intermediate page, use a link that has the Carry State, Display behavior from each originating node to the node for your intermediate page. Use a “Carry State, Create Report” link from the intermediate node to the destination node.

When you use the navigation nodes and links that have been supplied as part of a topic master, all of these transitions are configured correctly. Whenever you add new nodes to a topic, be sure to provide the appropriate intermediate Web pages and links for the transition from each origin to the node that you add.

The List Manager and Campaign Manager Web pages require that you provide intermediate pages whenever you wish to carry state from any of the reporting and analysis pages (other than Scoring), even across links with the Display behavior component. Other pages require intermediate pages only when a link specifies the Create Report behavior component.

Table 16 lists the destination Web pages for which intermediate nodes might be required. This table also shows the type of intermediate Web page that is required from each Web page type.

TABLE 16: INTERMEDIATE WEB-PAGE TYPES

(1 of 2)

	Destination Web Page Type	Intermediate Web Page Type	Origin Web Page Types (by row number in this table)
1	Advanced Rows and Columns Basic Rows and Columns	Add FOAM to Rows and Columns	3, 6, 7, 9, 13, 15
2	Campaign Calendar	n/a	n/a
3	Community Clusters	Add FOAM to Clusters	6, 9, 15
4	Cumulative Projections	Add FOAM to Rows and Columns	3, 6, 7, 9, 13, 15 (strongly recommended for 1, 11, 16)
5	Group Campaigns	Select Navigation Path to List Manager, Select Navigation Path to List Manager Popup Top N Scores	1, 3, 4, 7, 11, 13, 16 15

TABLE 16: INTERMEDIATE WEB-PAGE TYPES

(2 of 2)

	Destination Web Page Type	Intermediate Web Page Type	Origin Web Page Types (by row number in this table)
6	Group List Manager	Select Navigation Path to List Manager, Select Navigation Path to List Manager Popup	1, 3, 4, 7, 11, 13, 16
7	High/Low Clusters	Add FOAM to High/Low Clusters	6, 9, 15
8	Individual Campaigns	Select Navigation Path to List Manager, Select Navigation Path to List Manager Popup	1, 3, 4, 7, 11, 13, 16
		Top N Scores	15
9	Individual List Manager	Select Navigation Path to List Manager, Select Navigation Path to List Manager Popup	1, 3, 4, 7, 11, 13, 16
10	Influences	Add FOAM to Influences	6, 9, 15
11	Lifecycles	Add FOAM to Rows and Columns	3, 6, 7, 9, 17, 15 (strongly recommended for 1, 4, 16)
12	Modeling	Add FOAM to Influences	6, 9, 15
13	Profiling	Add FOAM to Profiles	6, 9, 15
14	Schedule	n/a	n/a
15	Scoring	n/a	n/a
16	Trends	Add FOAM to Trends	3, 6, 7, 9, 13, 15 (strongly recommended for 1, 4, 11)
17	View List	Add Transaction Type	1
	View List from Segment		

CONFIGURING NAVIGATION NODES FOR SCORING WEB PAGES

Follow these steps to configure navigation nodes that have the Scoring Web page type:

1. Create a Web Page of with the Scoring Web page type.
2. Select that Web page as the Web Page Contents for all navigation nodes with the Scoring Web page type.
3. Create at least two links in each Scoring navigation node:
 - one link in the New Model link category, to a node with a Modeling Web page, with the label “New Model.”
 - the other in the New List link category, to a node with a List Manager Web page, with the label “New List.”
4. E.piphany suggests adding additional links to additional nodes with other Web page types (such as Profiling) in the Analyze Further link category. These additional links allow users to perform additional analyses of data that they use for ranking list members.

CONFIGURING NODES FOR INFLUENCES, COMMUNITY CLUSTERS, AND MODELING WEB PAGES

You must ensure that the primary dimension used in any Influences, Community Clustering, or Modeling Web page that analyzes a list of individuals is the **indiv** dimension role. You must ensure that the primary dimension for any of these Web pages that analyzes a list of groups is the **group** dimension role.

CREATING INTERTOPIC LINKS

An intertopic link allows direct access from a navigation node in one topic to a navigation node in a different topic. Intertopic links can only be added in topics, not in topic masters. Aside from this, the process of adding intertopic links is the same as that for creating new links within a topic:

1. Edit the topic that contains the navigation node that is to be the origin of the intertopic link.
2. Click the **Add Link** button to display the Navigation Link dialog box. Select the destination node from the **Destination Node** drop-down list. This list box includes all of the navigation nodes in all of the topics that have been configured so far.

Note: When you save a topic to a topic master, all intertopic links are excluded from that topic master. When you create a new topic from that topic master, you must create new intertopic links.

PROVIDING ACCESS TO USERS AND GROUPS

You can give access to users and groups for specific nodes within a topic. This capability allows you to provide different views of the same topic to different users. If you do not give explicit access to a navigation node to a user or group, that person or group cannot view the Web page at that node.

Links from other navigation nodes to the node are also hidden from users and groups to whom you have not given access to a node. Links that originate at that node are also suppressed. This means that if a user or group does not have access to a node, and that node provides the only link to other nodes, the user or group is locked out of those nodes as well.

To provide access to an individual node:

1. Double-click the entry for that node in the Navigation Nodes tab of the Topic dialog box.
2. In the Access tab of the Navigation Node dialog box, click **Add Group**, then select the groups to whom you want to give access.

09625518.072500

3. To deny access, select groups from the **Groups** list box and click the **Remove** button for groups.
4. Select the users to whom you want to give access, then click **Add User**. To deny access, select groups from the **Users** list and click the **Remove** button.

To provide access to a number of Navigation nodes to one or more groups:

1. Select all of the applicable navigation nodes in the Navigation Nodes tab of the Topic dialog box.
2. Click **Add User** or **Add Group**.
3. Select from among the available list of users and groups in the Choose dialog box, then click **OK**.

ADDING NAVIGATION NODES

To add a navigation node to a topic, open the Navigation Nodes tab of the Topic dialog box for your topic, click the Add button, and then follow the instructions in the previous two sections for assigning Web pages and adding links.

HIDING OR DELETING NAVIGATION NODES

When you have completed the process of assigning Web pages and links to the nodes that your application currently needs, you have a choice about how to handle nodes that the topic master has included but that are not needed in the particular topic you are creating. One option is to delete each unused node by selecting it in the Navigation Nodes tab of the Topic dialog box and clicking **Remove**.

Another option is to hide unused nodes by removing access to them for users and groups. For ease of maintenance, E.piphany suggests hiding nodes rather than removing them.

To remove access to unused nodes:

1. Double-click the row for the node you want to hide in the Navigation Nodes tab of the Topic dialog box, or select the row and click **Edit**.
2. Open the Access tab in the **Navigation Node** dialog box.
3. Choose all groups from the **Group Name** list and click the **Remove** button for groups.
4. Choose all users from the **User Name** list box and click the **Remove** button for users.

Whether you hide or remove an unused node, links from that node to destination nodes are disabled. If the route to a node that you need passes through a node that you remove or hide, that node is rendered inaccessible to users. You must ensure that there is an alternate link to every node that is the destination of a link from the current node. This is, of course, much easier to do before you delete the node. If you hide the node instead, you can still view the destinations of its links in EpiCenter Manager, even though those links have been rendered inactive for end users.

CONFIGURING LIST-MANAGER NAVIGATION NODES

When you configure a node with a Group List Manager or Individual List Manager Web page, you must configure a link with a List Preview Web-page type. The Web page that you choose for the Web-page content at the destination link can be any Web page of the same Web-page type.

PROVIDING ON-LINE HELP FOR TOPICS AND WEB PAGES

Each E.piphany solution includes a set of on-line help pages. These Web-based pages describe the functionality of the various Web pages and give a generic overview of E.piphany solutions. E.piphany also provides a number of Web-based help pages that you must customize to provide site-specific information about the Web pages and topics that you configure, and the solution maps that you install:

- **Topics.html** and **Topic1.html**

You can customize the **Topics.html** file to display a list of the topics that you have configured, with links to help pages for each individual topic. The **Topic1.html** file is a template that you can copy and customize to create individual help pages for each of the topics that you have configured.

- **SN_*.html**

You can update these files to display local annotations to each of the standard help files that are part of your E.piphany application.

- **Solutions.html**

You can update this file to display a list of the E.piphany solution maps that have been installed as part of your application, with links to E.piphany-supplied help pages that describe these solution maps. The help pages for individual solution maps are installed along with the solution-specific software that you install.

These files reside in the following directory:

`C:\Program Files\E.piphany\Instance\WWWRoot\Help\SiteNotes`

Replace *C* with the drive on which your E.piphany software is installed and *Instance* with the name of your E.piphany instance. These files are in HTML format. E.piphany suggests that you use a text editor to copy, update, check, and then replace the raw HTML files, in order to avoid potential formatting problems that can be caused by some HTML authoring packages.

These help pages are available to all users of your E.piphany application. If you want to restrict access to information about a specific topic, you can use a node with a Navigation Path Web page as the beginning node of that topic, and enter the descriptive text as a prompt for that node.

CREATING AND EDITING TOPIC MASTERS

When you have completed a topic that has the right combination of navigation nodes, Web pages, and links, you can save it as a topic master by clicking the **Save As Master** button in the Topic dialog box. When you create a topic master from an existing topic, the new topic master includes the navigation nodes, Web page types, and links. The specific Web page contents are not included in the topic master.

To edit a topic master:

1. Open the **Solutions** folder in EpiCenter Manager, then the **Topic Master** folder, and then double-click the topic master you want to edit to display the Topic Master dialog box. The General tab of this dialog box displays the name of the topic master and the navigation nodes that the topic master includes.

Note: *Topic masters for E.piphany solution maps are not editable.*

2. Add, edit, or delete nodes as appropriate:
 - Use the **Add** button to add a node to the topic master. This button displays the Navigation Node dialog box (Figure 109, on page 339). Follow the instructions in “Configuring Navigation Nodes,” on page 337 to enter the appropriate information about the node itself, and the links you wish to assign to the node.

Note: *The Access and Reports tabs do not appear in the Navigation Nodes dialog box that is displayed for a topic master.*

09625518.072500

- Choose a node from the list and then click the **Edit** button (or double-click the node entry) to display the Navigation Node dialog box, which you can use to update information about the node and the links that have been assigned to it.
 - Choose a node from the list and then click the **Remove** button to delete that node and all of the links that originate at that node.
3. Choose a beginning node by selecting a node and checking the **Beginning** check box. Then choose a behavior for the link from the Home page to the beginning node. EpiPhany suggests that you choose the **Carry State, Display** behavior for this link.
 4. Enter a description for the topic master in the Description tab.

The Usage tab of the Topic Master dialog box displays the list of topics that have been derived from a topic master. You can update the topics that have been derived from the current topic master to reflect the edits that you have made in the topic master by selecting the each topic in EpiCenter Manager and clicking the **Update** button in the General tab of the Topic dialog box.

REFRESHING A TOPIC FROM AN UPDATED TOPIC MASTER

When you update a topic master, EpiCenter Manager does not propagate the changes that you have made in the topic master to existing topics that have been derived from that topic master. To refresh the structure of a topic from its updated topic master, double-click that topic in EpiCenter Manager, and then click the **Update** button in the Topic dialog box for that topic.

If the extent of the updates that you have made to a topic master reaches a level at which you would prefer to rebuild an existing topic rather than update it, you can delete the topic and then create it once again. However, whenever you delete a topic, EpiCenter Manager deletes all of the saved reports that are assigned to nodes within that topic.

Warning: *Deleting a topic deletes the saved reports that are assigned to nodes within that topic.*

You can preserve your save reports by exporting them, rebuilding your topic, and then importing them again, as follows:

1. Right-click the topic in EpiCenter Manager, then click **Export Saved Reports**.
2. Enter the name of a file in which to store the exported metadata for your saved reports in the Export dialog box, and then click **Go**.
3. Delete the topic.
4. Create a new topic that has the same name as the topic that you just deleted. If you want to rename the topic, do so after you have completed these steps.
5. Import saved reports from the file of exported metadata.

09625518.072500

THE E.PIPHANY APPLICATION SERVER

The E.piphany Application Server (AppServer) is the middleware component that allows users to interact with an EpiCenter datamart through the E.piphany Web-based user interface. The AppServer accepts requests that users submit with E.piphany Web pages, dispatches queries to the datamart, and returns the results of those queries to users in the form of Web pages that contain query results or responses.

Not all user requests require queries against the datamart. For instance, the following types of requests, among others, are processed by the AppServer itself:

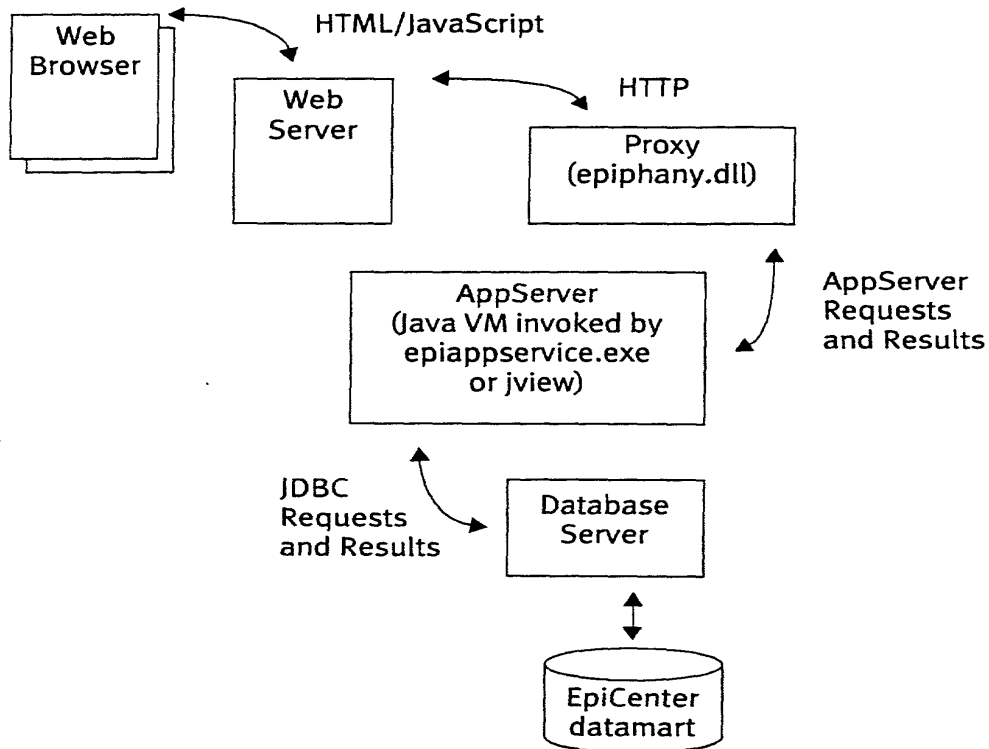
- Requests to log in to the E.piphany application
- Requests to display the Web page at a particular navigation node
- Requests to browse the Report Gallery

The AppServer is a Java application that manages user connections and database queries in a multithreaded fashion. Figure 112, on page 358, illustrates the role that the AppServer plays in the architecture of your E.piphany application.

The AppServer is implemented as a collection of Java classes that run within a Java virtual machine. The virtual machine that runs the AppServer code can be invoked as a Windows NT Server service, or directly as a **jview** command.

The AppServer connects to the Web server by means of a proxy program called **epiphany.dll**. The Web server passed HTTP requests to this proxy, which forwards those requests to the Java virtual machine that executes the AppServer code. The proxy isolates the AppServer from the Web server for greater reliability of your E.piphany application.

FIGURE 112: ROLE OF THE APPLICATION SERVER



An EpiPhany user session begins when a user requests the URL for an EpiPhany proxy as the destination to visit in a Web browser. The URL request is resolved by the Web server (IIS), and then routed to the EpiPhany proxy, **EpiPhany.dll**. The proxy is an Internet Server Application Programming Interface (ISAPI) application that packages the request and forwards it to the Application Server through a TCP/IP port.

The proxy handles requests by other users as it waits for the AppServer to process the request and return the result, which is the data to display the E.piphany login page. The proxy then forwards this result to the Web server that, in turn, routes that result to the user's Web browser for display. For more information about this proxy, refer to "The E.piphany Proxy," on page 380.

An E.piphany installation can include multiple Application Servers that each communicate over different TCP/IP ports. For example, a second Application Server might serve as a backup, or one Application Server might be used for development and another used for production.

For instructions on how to troubleshoot the Application Server, see Appendix H, "Troubleshooting." That appendix also describes Application-Server error messages.

STARTING AND STOPPING THE APPLICATION SERVER

The standard E.piphany software installation (as described in the *E.piphany e.4 Installation Guide*) configures the E.piphany Application Server as an NT service. You can start and stop the service using the Windows Control Panel\Services menu by following these steps:

1. From the Start menu, choose **Settings\Control Panel\Services**.
2. Select the E.piphany Application Server. The name of the service is the *instance_name* that you specified during installation.
3. Click Stop to stop the Application Server, or click Start to start it.

Tip: *The AppServer checks for inconsistencies in the metadata for your E.piphany application before it begins processing requests. Such inconsistencies can prevent the AppServer from starting successfully. The first step in diagnosing such a failure is to run the Scrutiny debugging tool. See "Running the Scrutiny Debugging Tool," on page 299, for details.*

MONITORING THE APPLICATION SERVER

To determine if the Application Server is running, enter the URL for the Epiphany proxy as the destination to visit in a Web browser. If the browser displays the Epiphany login page, the AppServer is running. If the Web browser displays an Epiphany proxy error message, you can attempt to start the AppServer by following the instructions in the previous section.

If the Web browser appears to stall or hang up, you can check the following resources to see if it is still running:

`http://www.hostname.com?monitor`

Replace *hostname* with the name of the computer on which the AppServer is supposed to be running.

1. Use the Web-based monitoring interface to query the AppServer. Enter a URL of the following form as the destination for your Web browser, as discussed in the next section
2. Check if the service is running in the Control Panel.
3. Check the NT Event log by going to **Start\Programs\Administrative Tools (common)\Event Viewer** and opening the **Log\Application** menu.

If the Service was started and is running, there will be an entry with **Source = EpiAppServer**. The message reads:

Epiphany Appserver instance message:
The Service was started.

Check for the existence of the AppServer log file. By default, this file resides in:

`C:\Program Files\Epiphany\instance\web\WWWRoot\logfiles`

or in the directory specified by the following registry key:

`HKEY_LOCAL_MACHINE\Software\Epiphany\Instances\
instance\SystemLogDir`

The log file has a name in the following date-time format. The filename indicates the year, month, day, hour, minute and second when the Application Server started running.

`YYYY-MM-DD_hh-mm-ss-32SRV.txt`

USING THE WEB-BASED APPSERVER MONITOR

Epiphany provides you with a Web-based interface for monitoring and refreshing the AppServer. This interface displays current status and activities, and allows you to browse the various logs that the AppServer creates. To log in to the AppServer monitor, supply the following URL as the destination for your Web browser:

`http://host/script/instance/epiphany.dll?monitor`

Replace *host* with the name of the host machine on which the AppServer runs. Replace *instance* with the instance name by which your E.piphany application has been installed.

Note: You must be a member of the Administrator group for your E.piphany application in order to log in to the AppServer monitor.

The AppServer monitor Web page is divided into the following topics:

- General
This section provides summary information about system resources that are currently being used by your E.piphany application.
- Current Activity
This section provides a link that you can use to view the current activity log.
- Statistics
This section provides a link that you can use to display statistics.
- Sessions
This section provides a link that you can use to view the session logs for individual users.

- Refresh

This section provides a link that you can use to refresh the AppServer.

- Debugging

This section provides a link that you can use to track threads.

The AppServer monitor allows you to examine current sessions and associated states. Click on a session to see a list of states that are currently loaded by the AppServer for that session. Click on a state to see the **Params** structure, a list of key-value pairs that describe the state.

REFRESHING THE APPLICATION SERVER

Upon startup, the Application Server reads metadata and Registry information and caches it for use during normal query processing. The metadata that is read includes:

- Information about the navigation nodes, link behaviors, Web pages, and Web-page templates
- Datamart configuration information
- The Windows Registry
- Aggregate navigation information.
- Time navigation information.
- Security information
- Storage information
- Scheduling information

When any of the following events occur, you need to restart or refresh the Application Server:

- After an extraction, either the **current_datamart** value has changed or there has been a change in a table that is used in a dynamic listbox filter
- If your EpiCenter datamart resides in an SQL Server database and the *MSSQL Server* service stops, you must restart both that service and the AppServer.
- EpiCenter Manager has been used to reconfigure your E.piphany application
- Aggregates have been built or rebuilt since the last time the Application Server was started
- The Windows Registry entries under the following entry have changed since the last time the Application Server was started:

HKEY_LOCAL_MACHINE/Software/Epiphany/*instance_name*

If you have determined that you need to refresh the Application Server, there are several ways to proceed. You can stop and then restart the Application Server via the Services Control Panel. However, this requires manual intervention, and is therefore unsuitable for programmatic refresh (after an extraction, for example). This method also interrupts anyone currently using the system.

Other ways to refresh the AppServer that do not interrupt users include:

- Using the **refresh.exe** command line
- Using the Web-based AppServer monitor. The URL for the AppServer monitor for an EpiPhany application takes the following form:

```
http://hostname/scripts/instance/Epiphany.dll?monitor.
```

Replace *hostname* with the name of the computer on which your E.piphany application resides. Replace *instance* with the name of your E.piphany application instance. To display the AppServer monitor, enter the URL in the **Address** or **Location** text box of your Web browser.

The AppServer monitor allows you to refresh the Application Server. It also allows you to monitor the current state and recent activity of the AppServer. To refresh the AppServer, click the **Refresh the Application Server** link. If the operation successful, the Web page displays the following message. Otherwise, the Web page displays an error message.

AppServer was refreshed

Refreshing does not interrupt the requests that are currently running on the Application Server.

Another way to refresh the server is by using the **refresh.exe** program that was installed in the **win32** subdirectory of your E.piphany installation directory. This program sends a message to the Application Server instructing it to reread all of the necessary information mentioned above.

You can use the **refresh** command to refresh the AppServer.

REFRESH

refresh *localhost port username password*

This command refreshes an E.piphany AppServer.

localhost	specifies the name of the machine on which the Application Server is running.
port	specifies the port number on which the Application Server is listening.
username	The <i>username</i> parameter specifies a valid administrative account. The
password	<i>password</i> parameter specifies a valid password for that account. These are the same values required for an end user to log into the top-level E.piphany Web page. In general, any account that is defined in the Security folder of the EpiCenter Manager application will work. See "Security," on page 273 for more information about E.piphany users.

After the **refresh** command has established a connection to the AppServer, it displays the following message and waits for a response:

Sent the REFRESH instruction to localhost

Normally, the Application Server takes about 30 seconds to refresh before returning an acknowledgment. (Times may vary based on the size of your EpiMeta database, the speed of your network, and other considerations.) Upon receiving this acknowledgment, the **refresh** program displays:

```
Refresh [Build 4.0.x.y]
-----
Connecting to localhost:8081
Sent the REFRESH instruction to localhost
The refresh SUCCEEDED.
REFRESH operation took 18827 ms.
```

Otherwise, **refresh** displays a failure indication. For example, here is the output of a negative interaction in which the user/password failed.

```
Refresh [Build 4.0.x.y]
-----
Connecting to localhost:8081
Sent the REFRESH instruction to localhost
The refresh FAILED because the username/password combination was
invalid.
REFRESH operation took 3365 ms.
```

The **refresh** program always displays the amount of time that it has taken.

MANAGING APPSERVER SERVICES

You can use the EpiAppService program (**epiappservice.exe**) to manage one or more AppServer services that run on the local host. This command allows you to perform the following actions:

- Add a new AppServer service
- Start or stop an AppServer service
- Change the parameters of an AppServer service
- Delete an AppServer service

The syntax of the **epiappservice** command follows.

EPIAPPSERVICE

(1 OF 2)

epiappservice -s -C -D -G -T -P -K [-DEP *svc*] [-E *cmd*] *servicename action*

This command registers an E.piphany AppServer as a service under Windows NT Server.

- S** specifies the name of the machine on which the Application Server is running.
- C** Creates a service. Registers a new service with the NT system. You must use the **-E** option with this option to specify the executable to use as the NT Service; for example:

```
EpiAppService -S "instance" -C -E "program name"
```

When you are configuring an E.piphany Application Server, the command line looks like this:

```
EpiAppService -S "instance" -C -E "C:\Program
Files\Epiphany\instance\win32\EpiAppService.exe -S
instance"
```

If the instance name has spaces in it, then you should use single quotes inside of the outside quotes. Do not use spaces within the instance name.

- D** Deletes the service specified via the **-S** option from the NT Service Registry. When this service is deleted, only the entry in the NT Registry for the service is deleted; no executables are deleted. Use this to remove unused Application Server instances.
- G** The launch handler for the service. (Go.)
- T** sTarts a service. The installation program uses this option to start the Web server (which must be shut down during installation). It starts the service specified via the **-S** option, which is equivalent to clicking the Start button in the **Control Panel\Services** dialog box to start the desired service. You may use this option when you cannot access the Control Panels (for example, during **RCMD** or **pcAnywhere** remote access).
- P** stoPs the service specified via the **-S** option. (See the **-T** option.) The installation program uses this option to stop the Web server during the installation.

EPIAPPSERVICE

(2 of 2)

- K** Checks if the service specified via the **-S** option is running. The program returns a zero return code if the service is running, and a nonzero return code if it is not.
- DEP** The *svc* argument specifies a service on which the Application Server depends.
- svc*** The Service Manager will always load this dependency before loading the Epiphany Application Server. The service name supplied as an argument to this option must exactly match the service name used by the service in question, such as Microsoft SQL Server or the Oracle listener.
- E *cmd*** The *cmd* argument specifies a command to associate with the service.
- ?** Displays online help, a description of the command options.

The following examples illustrate some common uses of the **epiappservice.exe** command:

- Creation of a service because the installation failed, or you wish to install another instance. If the database server is on a different machine from the Application Server:

```
EpiAppService -S instance -C -E "C:\Program Files\
Epiphany\instname\Win32\EpiAppService -S instname"
```

If the database server is on the same machine as the Application Server:

```
EpiAppService -S instance -C -E "C:\Program Files\
Epiphany\instance\Win32\EpiAppService -S instname" -DEP
MSSQLServer
```

- Deletion of an old service.

```
EpiAppService -S instance -D
```

RUNNING THE APPSERVER AS A CONSOLE APPLICATION

You can use the **jview** command to start the Application Server from a command console if you have the following Windows NT Server privileges:

- Act as part of OS
- Increase quotas
- Replace a process level token

Note: You can use the pass through security module without setting these privileges. See "Authentication Modules," on page 374.

The Local System account already has these privileges, and E.piphany recommends that you configure your Application Server to run as a service under the Local System account.

To run the Application Server from the command line, follow these steps:

Note: This procedure affects only the local machine, not the NT network.

1. Log in as a user who has local administration access.
2. Assign special NT privileges to a user account. To do so, start User Manager and enter the local machine name in the Select Domain dialog box.
3. Choose Policies\User Rights from the menu.
4. Click Show Advanced User Rights and assign the account of the currently logged-in user to have the privileges specified above.
5. Reboot your machine for the privileges to take effect.
6. In a command window, navigate to the following directory:

C:\Program Files\E.piphany\instance\classes

7. Enter **server** to run the **server.bat** file, or enter the following command to start the AppServer:

```
jview /cp:p C:\Program Files\E.piphany\instance\EpiAppServer.jar
com.epiphany.server.Server @instance
```

If the machine name and port number are not specified, then they are read from the Registry under the registry key for the instance that you have specified. If the *instance_name* is not specified, then the AppServer reads the default value in the following registry key to determine the instance name to use:

```
HKEY_LOCAL_MACHINE\Software\Epiphany\Instances
```

When you start the AppServer from the command line, log information is copied to a console window. This console window must remain open while the Application Server runs. If you close the console window, the Application Server terminates. If you log off, the console window closes, and the Application Server terminates.

APPLICATION SERVER LOGGING

Most components of the Application Server maintain a log file of their activities. This section describes each of these logs, their directory location, and their diagnostic use. In the Windows NT Registry on the Application Server machine, the Registry key that specifies the location of all log files generated by the Application Server is named:

```
HKEY_LOCAL_MACHINE\Software\Epiphany\Instances\instance_name
\SystemLogDir
```

All log file names begin with a prefix that indicates the date and time when the log file was first created. The format is as follows:

```
YYYY-MM-DD_HH-MM-SS-MStype.txt
```

YYYY stands for the year, *MM* for the month, *DD* for the day, *HH-MM-SS-MS* for the time (hour, minute, second, and millisecond), and *type* stands for log file type, as listed in Table 17, on page 370. The logs of queries for Epiphany applications have the suffix *QM_sessionid*. *Sessionid* stands for the session ID of the particular Web page from which the query originated.

TABLE 17: APPSERVER LOG TYPES

Log Type	Description
Server Log	Logs all connections made to the Application Server. This file contains the time at which a connection was accepted, the number of that connection, the time at which the request was finished, the total time required by the request, and certain messages printed to the log during the processing of that request. The latter category includes: the session ID of the connection, the template used to process the request, the number of bytes generated in the response, exceptions that might be generated, and the user name.
Connection Log	Logs all function calls, SQL, exceptions, and errors related to database-server connections.
Schedule Log	Logs all function calls, SQL, exceptions, and errors for the schedule subsystem.
SecurityManager Log	Logs all function calls, SQL, exceptions, and errors for the security subsystem.
StorageManager Log	Logs all function calls, SQL, exceptions, and errors for the storage subsystem.
Action Log	Each Web-page request generates a log that contains Web-page parameters, all of the SQL statements that have been executed during the processing of the request, and information about how long the processing took.

CACHING OF QUERY RESULTS

The AppServer caches results of queries that are run against the datamart. If another query uses the same dimensions, measures, and filters, the AppServer uses the data it has cached rather than issuing another query against the datamart. An optional registry key, **QueryCacheSize**, determines the number of query results that the AppServer caches. If this key is not set to another value, the AppServer caches the most recent 200 query results.

TRACKING OF QUERIES FOR AGGREGATE OPTIMIZATION

The AppServer maintains a log in metadata for the tables and columns that queries access. This log is used by the aggregate optimizer to generate statistics and make recommendations about which aggregates to build. You must run one of the following SQL scripts to clear these logs from time to time, in order to avoid filling the device or tablespace in which your EpiMeta database resides.

```
querylogtrunc_mssql.sql # for SQL Server
```

```
querylogtrunc_oracle.sql # for Oracle
```

These SQL scripts are located in the following directory:

```
C:\Program Files\Epiphany\instance\Configfiles
```

When you run this script, the data that the aggregate optimizer uses to make recommendations for new aggregates is removed. The next time you refresh the aggregate optimizer from the logs, all historical data prior to the time at which the script has been run is lost. Epiphany recommends that you refrain from refreshing the aggregate optimizer for an appropriate number of days after running this script to reestablish a history of recent queries.

APPLICATION SERVER SECURITY

Authentication, which is performed outside of the Epiphany system, does not capture password information. This external authentication requires an authentication module. For example, the NT authentication module authenticates users with an NT domain controller. The Security Manager inside the Application Server loads the authentication module specified through the SecurityClass Registry key at initialization. Each authentication module supports a fixed API that includes methods to authenticate users, add new users to the Epiphany system, and sync-up outside information on the user (such as group memberships) with Epiphany metadata.

Currently, EpiPhany provides two authentication modules: the NT authentication module EpiNTLogon, and an insecure authentication module called EpiPassThruLogon. Use EpiPassThruLogon only for testing and debugging. Optionally, you can add an LDAP authentication module, X.500 module, and other similar modules.

The optional Registry key SecurityClass under the *instance_name* Registry directory controls which security module is loaded. You must specify the full class path to the security module. If this key is omitted, the system uses the default security module, **com.epiphany.security.EpiNTLogon**, which uses NT to perform user authentication.

The means by which the authentication information (username and password) reaches the Application Server is as follows. Users log into the EpiPhany system either through a login template or through a Web server authentication mechanism, such as Basic Authentication or NTLM.

When the Application Server receives the user name and password, it calls the Security Manager to log in the user. The Security Manager loads an authentication module, and attempts the login process. If the process is successful, depending on the authentication module, one of the following steps is taken:

- If the user exists in the EpiMeta database, user group memberships outside of the EpiPhany system will be synched up with group memberships in the EpiMeta database.
- If the user does *not* exist in the EpiMeta database, but is authorized to use the EpiPhany system, then a user account will be created in the EpiMeta database. User group memberships outside of the EpiPhany system, such as NT, will be synched-up with group memberships in the EpiMeta database. For example, assume your EpiMeta had a group named EPIPHANYUSERS configured with access to all of the Web Pages. If a user name Joe (a valid NT user who belongs to the NT group EPIPHANYUSERS) supplies a correct password, then Joe will be automatically added to the EpiMeta metadata as a user who belongs to the EpiPhany group EPIPHANYUSERS.

Only group memberships for a group whose Group Definitions dialog box in EpiCenter Manager has the Synchronize option selected will be synced-up. Synchronization occurs when:

- The user who logs in is a member of a Group X outside of the E.piphany system, and:
 - a) that same group is defined within the E.piphany system
 - b) the user is not a member of that group inside the E.piphany system.
 In this case, sync-up adds a group membership to the E.piphany system.
- The user who logs in is a member of a Group X inside the E.piphany system, but the user is *not* a member of this group outside of the E.piphany system.
 In this case, sync-up removes a group membership from the E.piphany system.

If the sync-up process requires a creation of a new group membership for a user, certain access rights are set up based on this membership. The ability to save queries has the access rights of Save Group/Default, and global-level save access is Inherit. (See “Security,” on page 273 for more information.)

For a group to be the same in EpiMeta and the NT domain, it needs to have the identical name (case sensitive) in both. The group name in the NT domain includes the domain name, such as **Epiphany\Sarah**. You need to make sure that the group name in the EpiMeta includes the domain name for that group.

When the user is authenticated and user information is synced-up, Security Manager determines if the user is authorized to use the E.piphany system. The user is authorized when the password is correct and he or she belongs to at least one group in the E.piphany system.

AUTHENTICATION MODULES

The following authentication modules are included with the E.piphany software.

Module	Class Name
NT (default)	com.epiphany.security.EpiNTLogon
Pass through	com.epiphany.security.EpiPassThruLogon

- **NT (default)**
NT domain authentication. NT groups are imported into the E.piphany system through EpiCenter Manager. As mentioned, these groups need to have the Synchronize option selected in the Group Definition dialog box, which means that memberships to those groups will be synced-up. Users who belong to those groups can log into the E.piphany system. When a user logs in, his or her NT group memberships are synced-up to the EpiMeta database. It is also possible to create E.piphany-only groups inside EpiCenter Manager by omitting the Synchronize option. Thus, the E.piphany administrator can create arbitrarily complex permission hierarchies using EpiCenter Manager, independent of the manner in which NT domain security is set up.
- **Pass through** (*for in-house debugging purposes only; should never be used at a customer site after the initial E.piphany system setup*)
This is an insecure authentication that has no password. This is an E.piphany-only development module that ignores NT authentication altogether. You do not need passwords in this model: specify your user name exactly as it appears in EpiCenter Manager (it must include a domain name if such exists), and you will be logged in. There is no sync-up process. That means that an authenticated user that does not exist in the EpiMeta database will not be allowed to use the E.piphany system. E.piphany groups and users are created and managed through EpiCenter Manager.

AUTHENTICATION MODULE TIPS

- Users that do not belong to any groups in the E.piphany system are not allowed to log in. An error message that says that user is authenticated but not authorized to use E.piphany system is displayed whenever an unauthorized but NT-authenticated user logs into the Application Server. User memberships may be adjusted upon login if the user is a member of synchronized groups in E.piphany system. A user must be a member of at least one E.piphany group after the synchronization process completes.
- The optional Registry key **SecurityClass** (located in the *instance_name* Registry directory) controls which security authentication module is loaded. The full class path to the security module must be specified as the value for this key. If this key is omitted, the default security module is **com.epiphany.security.EpiNTLogon**, which uses NT to perform user authentication.
- **Toplevel.template** is the template that appears immediately after a user logs into the E.piphany system. The name of this template is configurable with the an optional ToplevelTemplate Registry entry in the **Instances** directory. You can load **company.template** instead of **toplevel.template** by substituting your company name in **ToplevelTemplate=company**.
- Depending on how IIS security is set up, one of the following situations occurs upon login:
 - If Allow Anonymous authentication is enabled, the login dialog box is displayed when the user attempts to use the system for the first time, or the user session times out. It is almost always sufficient to specify the username only. The domain name is located automatically.
 - The search order that authentication uses to find the user account is as follows. First, the authentication mechanism looks in the local machine's Security Access Manager (SAM), then it checks with the primary domain controller, and afterwards checks with trusted domains. If there are multiple users with the same name between domains and/or a local machine that runs the Application Server, specify the full user *name-domainname\username*, or the *local_machine_name\username* if the user logs in from a local machine's Security Access Manager (SAM).

- If Basic Authentication is enabled, the browser displays a login dialog box when the user attempts to access the E.piphany system for the first time. However, when the user's session times out, no re-login is required. The user is automatically logged in again, and a new user session created.
- If NTLM authentication is enabled, Internet Explorer automatically performs authentication of the user without displaying a login dialog box, although Netscape Navigator displays it. If Basic Authentication or NTLM is on, the login dialog box does not appear in the Web browser.

Warning: Do not create your own domains. Doing so introduces multiple domains, with the E.piphany machine in one domain and the user accounts of the E.piphany system in another domain. In order for the authentication module `com.epiphany.security.EpiNTLogon` to work properly, a two-way domain trust between the E.piphany domain and the customer domain is required.

If you set up a new domain for the machine that runs the E.piphany Application Server, set up a two-way trust and name the machine and the domain differently. In general, do not use the same string for domain names, machine names, and user names.

- The following information applies to Synchronized groups:
EpiNTLogon requires the NT domain for lists of global and local groups. The names of these groups will be matched to the names of the groups in the EpiMeta. A group name will be matched if its domain name and group name match. Therefore, group `xyz` will not match to group `EPIPHANY\xyz`. The match is case insensitive for both group name and domain name. If a user is a member of an NT group `EPIPHANY\xyz` and EpiMeta has a group called `epiphany\XYZ`, a match occurs.
- If a user is a member of a local group and the group has a global group as a member, the global group will not be picked for the synchronization process. Only the groups for which the user is an immediate member are considered for synchronization.

- If a user logs in with an account that is local to the Application Server machine, then a membership to a special group called None is automatically retrieved from the machine's SAM. (Every user in the local SAM has a membership in a special global group called None although this group does not exist on the machine.) For this reason, do not create synchronized groups called None in EpiCenter Manager.
- Avoid having the same name for domain names, machine names, and user names. For example, if the Application Server runs on the machine xyz, and the user called xyz attempts to log in, access may be denied. If the Application Server is running on a machine named xyz, and a user named xyz logs in from the primary domain, or from the local Security Access Manager (SAM) of the machine xyz, authentication will succeed. If user xyz logs in from a trust domain, however, the authentication will fail. The only way to log in as xyz from another domain is to give the full name for the user account upon login: *domainname\xyz*.

ADMINISTRATOR GROUPS

Epicenter Manager users can make any group an administrator group, and there can be multiple administrator groups in the system. If the user belongs to such a group, that user has special powers when it comes to report/folder and Web-page access.

Administrator users can save, overwrite, create, and change properties and permissions on any nonspecial and nonhidden file or folder.

Note: Special folders are top-level folders, such as Public and All Users, or user/group folders, or default folders. Hidden folders are the MailTo folder. Hidden files are clipboard files.

Administrator users have access to all Web Pages in the system.

APPLICATION SERVER REGISTRY KEYS

All Registry keys used by the E.piphany Application Server are located in:

HKEY_LOCAL_MACHINE/Software/Epiphany/Instances/*instance_name*

where *instance_name* specifies the name of the instance you entered during the Epiphany software installation. Table 18 lists the registry keys in this directory:

TABLE 18: APPSERVER REGISTRY KEYS

(1 OF 3)

Registry Key	Description
AppServerHost	The name of the machine on which the Application Server is running The default host is the local host.
AppServerLogVerbosity	The verbosity level for database logging performed by the AppServer If the value is 0, no logging is performed. If the value is 1, logging occurs.
AppServerPort	The port number on which the Application Server is listening for connections The default port is 8081.
AppServerQueryTimeout	The default timeout value, in seconds, before the Application Server automatically terminates long-running queries A query that exceeds the indicated number of seconds is terminated in order to prevent undue resource contention. This value can be overridden by a user, for her or his queries only, in the User Preferences dialog of the Home page.
AppSessionTimeout	The value in seconds for the lifetime of an idle session A session that remains idle for longer than this time is removed from the cache.
AppStateTimeout	The threshold, in seconds, at which Web-page state information is swapped from AppServer memory to disk
ChartsOutputDir	The directory (full path) in which the *.epc chart files will be stored

TABLE 18: APPSERVER REGISTRY KEYS

(2 OF 3)

Registry Key	Description
DatabaseName	The name of the EpiMeta database
DatabasePassword	The password that corresponds to the DatabaseUsername account
DatabasePort	The port that supports the connection to the database server The default port for SQL Server is 1433. The default port for Oracle is 1521.
DatabaseServer	The name of the database server on which the EpiMeta database is located
DatabaseType	The type of the database
DatabaseUsername	A username for logging into both the EpiMeta and EpiMart databases
Description	A textual description of the instance
InstanceRootDir	The root directory into which the EpiPhany Application Server has been installed
MaxDBConnections	The maximum number of simultaneous open connections to the database server. The default number is 100
MomentumOutputDir	The directory in which campaign output files are stored
ProxyLogFile	(Optional) Enables proxy logging For example, setting this key to C:\proxy.log will create a proxy.log file if it does not already exist, and log information for every proxy submit to the Application Server. If an invalid path is specified in this ProxyLogFile Registry key, no logging will be performed.
QueryCacheSize	The number of most-recent query results to store in the AppServer cache
ScrutinyDisabled	(Optional) When present and set to Yes , disables the Scrutiny checks on metadata that are automatically performed when the AppServer starts up

TABLE 18: APPSERVER REGISTRY KEYS

(3 OF 3)

Registry Key	Description
SecurityClass	(Optional) The Java class to use for security authentication The default class is com.epiphany.security.EpiNTLogon .
StateDir	The directory in which Web-page state information is temporarily stored
SystemLogDir	The directory in which the Epiphany Application Server log files are written
SystemLogDirWebpath	The name appended to http://machinename/instance_name/ that informs the Web server of the locations of log files.
TempDirGarbageLifetime	The lifetime in seconds of a temporary or log file created by the Epiphany Application Server After this specified number of seconds from the creation date, a temporary file, log file, or *.epc chart file will be erased by the Temporary File Manager in the Application Server. Garbage collection occurs at Application Server startup and periodically when there are free cycles.
TemplateDir	The directory in which the Epiphany Application Server templates are stored
WWWRootDir	(Optional) The path to the Epiphany instance Web directory, for example: C:\Program Files\Epiphany\ instance\Web\WWWroot

THE E.PIPHANY PROXY

The Epiphany proxy (**Epiphany.dll**) is an ISAPI application that mediates the requests and responses between the IIS Web server and the Epiphany Application Server. All user requests for Epiphany pages are directed to the proxy:

```
http://machinename/scripts/instance_name/Epiphany.dll
```

This proxy bundles the request into a package that conforms to a strict E.piphany format and sends the package to the E.piphany Application server through a TCP/IP socket. The proxy uses the Windows Registry to find the E.piphany Application Server. In particular, the proxy parses the *instance_name* from the requesting URL. It then opens the AppServerHost and AppServerPort found in the Registry tree for that instance, and uses this information to connect to the Application Server. The Application Server processes the request and sends a result back to the proxy. The proxy displays the result in the user's browser.

Note: The instance_name in the URL must match the instance_name as defined in the Windows Registry (configured from the installation program). This allows one proxy to direct requests to several different Application Server instances. This is of value when you want to have two instances, such as a release and a test instance.

PROXY LOGGING

The **Epiphany.dll** proxy supports rudimentary logging. If you suspect that the proxy is not passing all parameters to the Application Server, you can enable logging of all parameters that the proxy submits.

Proxy logging, which is optional, is a diagnostic tool for identifying problems, not a run-time logging facility. Because the log file may grow arbitrarily large if proxy logging is always enabled, use it only in the event of a suspected problem with the **Epiphany.dll** proxy.

An example of a proxy log file:

```
Processing new request. Header: mGET q p v0.8 oHTTP/1.0 nzhenya P80
r192.0.0.147 aGodzilla/4.04 [en] (WinNT; I ;Nav) u
U/scripts/capri/Epiphany.dll S/scripts/capri/Epiphany.dll
Requested dispatched. Request data length was 0
```

```
Processing new request.
...
Request data length was 72
```

The log file consists of blocks, with each block representing a request to the proxy DLL. Each new request starts with a `Processing new request` message, and then a request header. The header is followed by the data section if the request was of the type `POST`. First, a number of bytes is printed (172), then the actual data. Finally, if the request was successful, the log entry ends with the following message:

```
Requested dispatched. Request data ....
```

If a request fails, the entry ends with the following message:

```
FAILED TO DISPATCH REQUEST DUE TO A SOCKET ERROR.
```

PROXY REDIRECTION

The Epiphany proxy allows you to set up automatic redirection to another URL. This may be desirable when you would like to move users from an old AppServer host machine to another host. The AppServer need not be running on the old machine, so long as the proxy remains installed. To configure the proxy to redirect access to the URL for a new AppServer, enter the following registry key:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Epiphany\Instances\instance\  
RedirectURL
```


E.PIPHANY MACROS

This appendix describes the built-in macros for SQL and operating-system commands that E.piphany supplies. You can use E.piphany macros in extraction jobs, queries against your EpiCenter datamart, and the command line for the AppServer. You can also define your own macros with EpiCenter Manager. Refer to "Macros," on page 241 for details on creating your own E.piphany macros.

BUILT-IN SQL MACROS

E.piphany provides a number of built-in SQL macros that allow you to write extraction jobs and queries against your datamart in a database-independent fashion. Where necessary, E.piphany also provides certain database-specific macros.

SQL MACRO SYNTAX

The syntax for an E.piphany SQL macro takes the following general form. Macro references begin with a pair of dollar signs. Arguments to SQL macros are enclosed within square brackets. The separator for arguments is a comma surrounded by tilde characters.

SQL MACRO SYNTAX

`$$MACRO`**`$$MACRO[argument]`****`$$MACRO[arg1~,~arg2]`**

White space between the macro reference and the opening square bracket of the argument list is not allowed. The following example expands correctly:

`$$NVL[MAX(col_1)~,~0]`

The following example does *not* expand correctly:

`$$NVL [MAX(col_1)~,~0]`

The E.piphany macro interpreter allows white space between arguments and argument separators. However, those white-space characters are passed through and appear as part of the expanded SQL statement. Take care to ensure that any white-space characters you embed in your arguments list do not adversely affect the resulting SQL syntax.

You can determine the expanded value for most E.piphany macros by issuing the following SQL command in an EpiMeta database:

`Select * from translation_actual`

For usage examples, see the initialization file **templates.sql** in the following folder:

`C:\Program Files\Epiphany\instance\ConfigFiles`

Replace *C* with the drive on which your E.piphany software is installed. Replace *instance* with the name of your E.piphany instance.

DATABASE-INDEPENDENT MACROS

EpiPhany supports multiple database servers for EpiCenter datamarts. The EpiPhany database-independent macros allow you to isolate your datamart from syntax differences that result from vendor-specific extensions to SQL.

Database-independent macros are classified into the following groups:

- Extraction macros
- EpiCenter-management macros
- General-purpose macros

Each group is discussed in a section that follows.

EXTRACTION MACROS

The extraction macros listed in Table 19 identify source-system data elements such as tables and columns, or destination data elements. The Usage column indicates the expected frequency of use for each macro.

TABLE 19: EXTRACTION-SET IDENTIFICATION MACROS

(1 OF 4)

Macro	Usage	Description
<code>\$\$COLUMN_CURRENT_VALUE[<i>table_name</i> ~,~ <i>column_name</i>]</code>	Low	Expands to the value of the indicated column as of the start of the current run. Can be used to complete a “two-sided” WHERE clause that also uses the <code>\$\$COLUMN_RANGE_FILTER</code> macro.
<code>\$\$COLUMN_FILTER[<i>table_name</i> ~,~ <i>column_name</i> ~,~ <i>alias_name</i>]</code>	High	Expands to a “one-sided” SQL comparison expression that requires that <i>alias_name.column_name</i> be greater than or equal to the value in table <i>table_name</i> column <i>column_name</i> as of the start of the last run. This extracts “everything since last time.” Although <i>alias_name</i> is optional, EpiPhany suggests that you include it.

TABLE 19: EXTRACTION-SET IDENTIFICATION MACROS

(2 OF 4)

Macro	Usage	Description
<code>\$\$COLUMN_LAST_VALUE[<i>table_name</i> ~,~ <i>column_name</i>]</code>	High	Expands to the value of the indicated column as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
<code>\$\$COLUMN_RANGE_FILTER[<i>table_name</i> ~,~ <i>column_name</i> ~,~ <i>alias_name</i>]</code>	High	Expands to a “two-sided” SQL comparison expression that requires <i>alias_name.column_name</i> to be greater than or equal to the value in table <i>table_name</i> column <i>column_name</i> as of the start of the last run, and less than or equal to this value as of the start of the current run. This extracts “everything since last time, but not including that data that changes while the extractions are running.”
<code>\$\$DATE_CURRENT_VALUE</code>	Low	Expands to the “current date” as of the start of the current run. This expands to a value, not to an expression, allowing you to make your own expressions. Can be used to complete a “two-sided” WHERE clause that also uses the <code>\$\$DATE_RANGE_FILTER</code> macro.
<code>\$\$DATE_FILTER[<i>column_name</i>]</code>	High	Expands to a “one-sided” SQL comparison expression that requires the column to be greater than or equal to the “current date/time” as of the start of the last run. This extracts “everything since last time.” No table or alias arguments are available to the macro. If the column needs to be qualified, just add the qualifications in the first argument. For example: <code>\$\$DATE_FILTER[oo.date_key]</code>

TABLE 19: EXTRACTION-SET IDENTIFICATION MACROS

(3 OF 4)

Macro	Usage	Description
\$\$DATE_LAST_VALUE	High	<p>Expands to the “current date” as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.</p> <p>This compares SQL Server datetimes. The column used for the comparison must be declared as a datetime or some variant in SQL Server. Only the date portion of the value is used; the time portion is discarded.</p>
\$\$DATE_RANGE_FILTER[column_name]	High	<p>Expands to a “two-sided” SQL comparison expression that requires the column to be greater than or equal to the “current date/time” as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts “everything since last time, but not including that data that changes while the extractions are running.”</p> <p>This compares SQL Server datetimes. The column used for the comparison must be declared as a datetime or some variant in SQL Server. Only the date portion of the value is used; the time portion is discarded.</p>
\$\$YYYYMMDD_CURRENT_VALUE	Low	<p>Expands to the “current date in YYYYMMDD format” as of the start of the current run. This expands to a value, not to an expression, allowing you to make your own expressions.</p>
\$\$YYYYMMDD_FILTER[column_name]	High	<p>This is the same as a DATE_FILTER except that only the “day” portion of the date/times is used. (Some business semantics are most meaningful when applied to “days.”) This extracts “everything since the last day, including the last day.”</p>

TABLE 19: EXTRACTION-SET IDENTIFICATION MACROS

(4 of 4)

Macro	Usage	Description
\$\$YYYYMMDD_LAST_VALUE	High	Expands to the “current date in YYYYMMDD format” as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.
\$\$YYYYMMDD_RANGE_FILTER[column_name]	High	This is the same as a DATE_RANGE_FILTER except that only the “day” portion of the date/times is used. (Some business semantics are most meaningful when applied to “days.”) This extracts “everything since the last day, including the last day, but not including today.”

EPICENTER-MANAGEMENT MACROS

The EpiCenter-management macros listed in Table 20 specify data elements within your datamart or conditions that might be true with respect to your datamart.

TABLE 20: EPICENTER MACROS

(1 of 2)

Macros	Usage	Description
\$\$CURR	High	Expands to the __A or __B suffix of the currently active tables. Allows you to reference the active or new EpiMart tables.
\$\$CURREXP	High	Expands to the __P or __Q suffix of the currently active backfeed tables. Allows you to reference the active or new EpiMart tables.
\$\$CURRHIST	High	Expands to the __X or __Y suffix of the currently active history tables. Allows you to reference the active or new EpiMart tables.

TABLE 20: EPICENTER MACROS

(2 OF 2)

Macros	Usage	Description
\$\$CURRVIEW	High	Expands to the suffix (such as <code>_AX</code> or <code>_BY</code>) of the currently active combined view of tables and history tables. Allows you to reference the active or new EpiMart tables.
\$\$DEBUG	Low	If debugging is enabled: Expands to nothing if the verbosity level of the extract.exe command is less than 3; otherwise, returns its argument. Note: For testing that depends on whether debugging is on or off, use both <code>DEBUG</code> and <code>NOT_DEBUG</code> .
\$\$INITIAL_LOAD	Medium	Indicates that timestamps are to be ignored during an extraction job. (The action of this macro is not related to the Initial Load semantic type.)
\$\$MARTDBNAME	Medium	Returns the name of the datamart database (EpiMart).
\$\$METADBNAME	Medium	Returns the name of the metadata database (EpiMeta).
\$\$NEXT	High	Expands to the <code>_A</code> or <code>_B</code> suffix of the currently <i>inactive</i> tables.
\$\$NEXTEXP	High	Expands to the <code>_P</code> or <code>_Q</code> suffix of the currently <i>inactive</i> backfeed tables.
\$\$NEXTHIST	High	Expands to the <code>_X</code> or <code>_Y</code> suffix of the currently <i>inactive</i> history tables.
\$\$NOT_DEBUG	Low	If debugging is <i>not</i> enabled: Expands to a null value if the extract command's verbosity level is higher than 3; otherwise, returns its argument. Note: For testing that depends on whether debugging is on or off, use both <code>DEBUG</code> and <code>NOT_DEBUG</code> .
\$\$NOT_INITIAL_LOAD	Medium	Indicates that timestamps are to be considered during an extraction job. (The action of this macro is unrelated to the Initial Load semantic type.)

GENERAL-PURPOSE MACROS

The macros listed in Table 21 are used for a variety of purposes.

TABLE 21: GENERAL-PURPOSE SQL MACROS (1 of 18)

Macros	Usage	Description
<code>\$\$ADD_DAYS</code>	Medium	Returns a date representation of its first argument plus its second argument as a number of days: For example: <pre>SELECT \$\$ADD_DAYS [\$\$DBNOW~, -1]</pre>
<code>\$\$ADD_MACRO[<i>macro</i>~,~ <i>dbtype</i>~,~<i>value</i>]</code>	Low	Defines a macro or assigns a new value. The <i>macro</i> argument is the name of the macro. The <i>dbtype</i> argument is the database server for which the value argument applies. The <i>value</i> argument is the value that the macro expands to for the indicated database server.
<code>\$\$ADD_MONTHS[<i>date_expression</i>~,~ <i>number</i>]</code>	Medium	Takes two arguments; adds the second as a number of months to the first argument, which is a date.
<code>\$\$ASSERT_INDEX_EXISTS[<i>index_name</i>]</code>	Low	Causes an SQL error if the index named in argument 1 does not exist. For example: <pre>\$\$BEGIN_ASSERT_INDEX \$\$ASSERT_INDEX_EXISTS['XPKCustMap_B'] \$\$ASSERT_INDEX_EXISTS['XPKAppMap_B'] \$\$END_ASSERT_INDEX</pre>

TABLE 21: GENERAL-PURPOSE SQL MACROS

(2 OF 18)

Macros	Usage	Description
<code>\$\$BATCH_PARALLEL_DEGREE</code>	High	Sets the parallel degree for extraction jobs. The initial value is 4. You can change the value with EpiCenter Manager. See “Macros,” on page 241.
<code>\$\$BEGIN_ASSERT_INDEX</code>	Low	In Oracle, declares the <code>DECLARE INDEX_NOT_EXISTS</code> exception. See <code>\$\$ASSERT_INDEX_EXISTS</code> .
<code>\$\$BIGDATE</code>	Low	Declares a <code>BIGDATE</code> data type (to record millisecond-precision timestamps).
<code>\$\$BOOL_TO_YN[<i>testvalue</i>]</code>	Low	Returns the string <code>N</code> if <i>testvalue</i> equals 0. Otherwise, returns <code>Y</code> . For example: <code>\$\$BOOL_TO_YN[6]</code> becomes <code>Y</code> .
<code>\$\$CASE_BEGIN</code>	Medium	Begins a case statement that compares an expression to a list of options and returns the value for the first matching option. Also provides a single option-value pair. For example: <pre>SELECT \$\$CASE_BEGIN[col_name~, ~opt 1~, ~val1] \$\$CASE_ELSEIF[col_name ~, ~ opt2~, ~val2] \$\$CASE_ELSE[else_val] \$\$CASE _END</pre>
<code>\$\$CASE_ELSE</code>	Medium	Fall-through value for a case statement that compares an expression to a list of options and returns the value for the first matching option. See <code>CASE_BEGIN</code> .

TABLE 21: GENERAL-PURPOSE SQL MACROS

(3 of 18)

Macros	Usage	Description
\$\$CASE_ELSEIF	Medium	Continues a case statement that compares an expression to a list of options and returns the value for the first matching option. See \$\$CASE_BEGIN .
\$\$CASE_END	Medium	Ends a case statement that compares an expression to a list of options and returns the value for the first matching option. See \$\$CASE_BEGIN .
\$\$CAT	High	Used as an operator to append “two” \$\$CAT strings. For example: <pre> \$\$TO_CHAR[table1.col1] \$\$CAT '-' \$\$CAT \$\$TO_CHAR[col2]</pre>
\$\$CBIN_VAL[<i>testval</i> ~, ~ lowerbound ~, ~ upperbound ~, ~ binletter]	Medium	Can be used to “bin” numeric values into character buckets. Multiple \$\$CBIN_VAL macros should be followed by a single \$\$CBIN_END . If <i>testval</i> is in the range lowerbound to upperbound (inclusive), then the expression yields <i>binletter</i> . For example: <pre> \$\$CBIN_VAL[7~, -1~, -5~, ~'A'] \$\$CBIN_VAL[7~, -6~, -10~, ~'B'] \$\$\$CBIN_END</pre>
\$\$CBIN_END		returns the value B.
\$\$CHAR_1	Medium	Expands into a type definition for a single character field.

TABLE 21: GENERAL-PURPOSE SQL MACROS

(4 of 18)

Macros	Usage	Description
\$\$COUNT_ROWS_FROM \$\$COUNT_ROWS_SELECT	Low	Can be used to count the number of rows in a table. Uses sysindexes on SQL Server for fastest count. For example: <pre> SELECT \$\$COUNT_ROWS_SELECT the_count \$\$COUNT_ROWS_FROM[MyTable]</pre>
\$\$COUNTER[<i>initial_value</i>]	Low	Returns sequential row numbers for a result set, starting with <i>initial_value</i> , if supplied. For example: <pre> SELECT \$\$COUNTER the_counter.</pre>
\$\$CREATE_INDEX_IF_NOT_EXISTS[<i>index_type</i> ~,~ <i>index_name</i> ~,~ <i>table_name</i> ~,~ <i>column_list</i> ~,~ <i>after_creation_clause</i>]	Low	Creates an index if it is not already there. <pre> \$\$DDL_BEGIN \$\$CREATE_INDEX_IF_NOT_EXISTS[UNIQUE ~,-XPK_123~,- table1~,-ss_key,iss, date_key,transtype_key,seq- ,-] \$\$DDL_END</pre>
\$\$DBNOW	High	Returns the date/time from the database. For example: <pre> SELECT Coll ss_key \$\$DBNOW date_modified from zork</pre>

TABLE 21: GENERAL-PURPOSE SQL MACROS

(5 OF 18)

Macros	Usage	Description
<code>\$\$DDL_BEGIN</code>	Low	<p>Starts a block of code that changes the schema. Use outside a DECLARE block. For example:</p> <pre>\$\$DDL_BEGIN \$\$NOT_DEBUG[\$\$DROP_TABLE_IF_EXISTS[table]] \$\$DDL_END</pre>
<code>\$\$DDL_BEGIN_NO_DECLARE</code>	Low	<p>Starts a block of code that changes the schema. Use inside a DECLARE block. For example:</p> <pre>DECLARE \$\$VAR[txnFIXED] \$\$VARCHAR_50\$\$EOS \$\$DDL_BEGIN_NO_DECLARE \$\$VAR_ASSIGN_BEGIN[txnFIXED] SELECT \$\$TO_CHAR[transtype_key] \$\$VAR_ASSIGN_INT0[txnFIXED] FROM Transtype_O WHERE name = 'FINV_ADJUST' \$\$VAR_ASSIGN_END</pre>
<code>\$\$DDL_END</code>	Low	<p>Ends a block of SQL that changes the schema. For example:</p> <pre>\$\$DDL_BEGIN \$\$DROP_TABLE_IF_EXISTS[\$\$FCTTBL[]\$\$NEXT] \$\$DROP_TABLE_IF_EXISTS[\$\$FCTTBL[]_INC] \$\$DDL_END</pre>

TABLE 21: GENERAL-PURPOSE SQL MACROS

(6 OF 18)

Macros	Usage	Description
\$\$DDL_EXEC[<i>statement</i>]	Low	<p>All items in the argument list are evaluated at runtime, not when the statement is parsed. This macro can construct SQL based on the values of variables computed in the same SQL block. For example:</p> <pre> \$\$DDL_BEGIN \$\$DDL_EXEC[CREATE INDEX X_table1 ON table1 (iss, ss_key, date_key)] \$\$DDL_END </pre>
\$\$DECLARE_BEGIN	Low	<p>Starts a DECLARE block. For example:</p> <pre> \$\$DECLARE_BEGIN \$\$DECLARE_BODY[\$\$VAR[count_INC] \$\$EPIINT] \$\$DECLARE_BODY[\$\$VAR[count_FC] \$\$EPIINT] BEGIN \$\$VAR_ASSIGN_BEGIN[cnt_INC] SELECT COUNT(1) \$\$VAR_ASSIGN_INTO[cnt_INC] FROM \$\$FCTTBL[]_INC \$\$VAR_ASSIGN_END </pre>
\$\$DECLARE_BODY[<i>argument</i>]	Low	Treats its argument as a declaration. See \$\$DECLARE_BEGIN .
\$\$DOUBLESTRING	Low	Expands to a VARCHAR data type that is used for decimal values in the campaign manager. This string type eliminates null values, which are not allowed in dimension tables.

TABLE 21: GENERAL-PURPOSE SQL MACROS

(7 OF 18)

Macros	Usage	Description
<code>\$\$DROP_INDEX[<i>table_name</i>~,~ <i>index_name</i>]</code>	Medium	Drops the index. For example: <pre> \$\$DDL_BEGIN \$\$DROP_INDEX[table1~,~ index_name] \$\$DDL_END </pre>
<code>\$\$DROP_TABLE_IF_EXISTS[<i>table_name</i>]</code>	Medium	Drops the table without returning an error indicating that the table does not exist. For example: <pre> \$\$DDL_BEGIN \$\$DROP_TABLE_IF_EXISTS[tbl1] \$\$DROP_TABLE_IF_EXISTS[tbl2] \$\$DDL_END </pre>
<code>\$\$ELSE</code>	Medium	The start of the negative clause of an IF statement.
<code>\$\$END_ASSERT_INDEX</code>	Low	Ends a block of checks that indexes exist. See <code>\$\$ASSERT_INDEX</code> .
<code>\$\$END_IF</code>	Medium	Ends an IF statement, see <code>\$\$IF</code> .
<code>\$\$EOS</code>	Low	Ends an SQL statement. For example: <pre> SELECT 'PROCESSED', COUNT(1), 1100 FROM table1 \$\$EOS </pre>
<code>\$\$EPIINT</code>	Medium	Declares an integer. For example: <pre> \$\$DECLARE_BEGIN \$\$DECLARE_BODY[\$\$VAR[unjoined] \$\$EPIINT] \$\$DECLARE_BODY[\$\$VAR[processed] \$\$EPIINT] </pre>

TABLE 21: GENERAL-PURPOSE SQL MACROS

(8 OF 18)

Macros	Usage	Description
\$\$EPIKEY	Medium	Declares an E.piphany dimension key. See \$\$EPIINT .
\$\$EXEC_SP [<i>proc</i> ~,~ <i>params</i>]	Low	Execute a stored procedure. The <i>proc</i> argument is the procedure name. The <i>params</i> argument is a comma-separated list of parameters to the stored procedure.
\$\$FACTMONEY	Medium	Declares a monetary value. See \$\$EPIINT .
\$\$FACTQTY	Medium	Declares a decimal value. See \$\$EPIINT .
\$\$FLOAT	Medium	Declares a float value. See \$\$EPIINT .
\$\$IDENTITY	Medium	Declares a integer serial sequence. See \$\$EPIINT .
\$\$IF [<i>condition</i>]	Medium	Performs a conditional action. For example: <pre> \$\$IF[\$\$VAR[fc_exists] = 0] \$\$DDL_EXEC[\$\$SELECT_INTO_BEGIN[tmp_tbl] SELECT * \$\$SELECT_INTO_BODY[tmp_tbl] FROM Old_table WHERE 1=0] \$\$END_IF \$\$DDL_END </pre>
\$\$I)_FROM [<i>table_name1</i> ~,~ <i>table_name2</i>]	Low	Performs an inner join on two tables. See \$\$JOIN_WHERE .

TABLE 21: GENERAL-PURPOSE SQL MACROS

(9 OF 18)

Macros	Usage	Description
<code>\$\$INSTR[<i>s1</i>~,~<i>s2</i>]</code>	Medium	Returns the position of <i>s1</i> in <i>s2</i> . For example: <code>\$\$INSTR['b' ~,~ 'abc']</code> returns 2.
<code>\$\$INTERACTIVE_PARALLEL_DEGREE</code>	Medium	Sets the parallel degree for ad-hoc user queries. The initial value is 4. You can change the value with EpiCenter Manager.
<code>\$\$JOIN_LEFT_OUTER</code>	Medium	Produces a condition for outer joining the first argument to the second. For example: <code>SELECT ... WHERE \$\$JOIN_LEFT_OUTER[t1.c1 ~,~ t2.c2]</code>
<code>\$\$JOIN_RIGHT_OUTER</code>	Medium	Produces an equals sign appropriate for right outer joins (No arguments are needed.) For example: <code>SELECT ... WHERE t1.c1 \$\$JOIN_RIGHT_OUTER t2.c2</code>

TABLE 21: GENERAL-PURPOSE SQL MACROS

(10 OF 18)

Macros	Usage	Description
\$\$JOIN_WHERE[<i>join_condition</i>]	Low	<p>Supplies the WHERE clause for a join. For example:</p> <pre> SELECT col1, col2 FROM Table1 s \$\$LOJ_FROM[table2 m,-s.iss = m.iss AND s.col2=m.col2] \$\$LOJ_FROM[table3 d,-m.col1 = d.col1] WHERE 1=1 \$\$JOIN_WHERE[m.col1=d.col1(+)] \$\$JOIN_WHERE[s.iss = m.iss (+) AND s.col2 = m.col2 (+)] </pre>
\$\$LENGTH[s]	Medium	<p>Returns the length of a string. For example:</p> <pre> \$\$LENGTH['abc'] </pre> <p>returns 3.</p>
\$\$LOJ_FROM[<i>join_condition</i>]	Low	Performs a left outer join. See \$\$JOIN_WHERE .
\$\$LONGSTRING	Low	Expands to a VARCHAR data type that is used for decimal values in the campaign manager. This string type eliminates null values, which are not allowed in dimension tables.
\$\$MAX_SYS_DATE	Medium	Returns the highest date supported by the database.
\$\$METADBNAME	Medium	Returns the name of the current metadata database (SQL Server) or schema (Oracle).

TABLE 21: GENERAL-PURPOSE SQL MACROS

(11 OF 18)

Macros	Usage	Description
\$\$MODULO [<i>x</i> ~, ~ <i>y</i>]	Low	Returns the remainder when <i>x</i> is divided by <i>y</i> . For example: <code>MODULO{ 7~, -4 }</code> returns 3.
\$\$MONEYSTRING	Low	Expands to a VARCHAR data type that is used for decimal values in the campaign manager. This string type eliminates null values, which are not allowed in dimension tables.
\$\$NBIN_VAL [<i>testval</i> ~, ~ <i>lowerbound</i> ~, ~ <i>upperbound</i> ~, ~ <i>binnumber</i>]	Medium	Can be used to “bin” numeric values into numeric buckets. Multiple \$\$NBIN_VAL macros should be followed by a single \$\$NBIN_END. If <i>testval</i> is in the range <i>lowerbound</i> to <i>upperbound</i> (inclusive), then the expression yields <i>binnumber</i> . For example: <code>\$\$NBIN_VAL[7~, -1~, -5~, -1]</code> <code>\$\$NBIN_VAL[7~, -6~, -10~, - 2]</code> <code>\$\$NBIN_END</code> return the value 2.
\$\$NBIN_END		
\$\$NO_FROM_LIST	Medium	Supplies the “dummy” FROM clause needed by some database vendors. For example: <code>SELECT 'MODIFIED',</code> <code>\$\$VAR[modified], 1050</code> <code>\$\$NO_FROM_LIST\$\$EOS</code>
\$\$NUMBER (<i>g</i> , <i>5</i>)	Medium	Declares a decimal(<i>g</i> , <i>5</i>). See \$\$EPIINT.

TABLE 21: GENERAL-PURPOSE SQL MACROS

(12 OF 18)

Macros	Usage	Description
\$\$NVL[<i>expression</i>~,~<i>value</i>]	High	When the first argument is NULL, replace it with the value in the second argument. For example: <pre>SELECT \$\$TO_CHAR[\$\$NVL[MAX (col1) ~ , ~1]]</pre>
\$\$ORACLE[<i>expression</i>]	High	Expands to nothing if the database is not Oracle. For example: <pre>SELECT COUNT (1) FROM \$\$SQLSERVER[sysobjects] \$\$OR ACLE[tabs]</pre>
\$\$RAISE_EXCEPTION[<i>exception</i>]	Low	Raises the given exception (as a variable on Oracle, as a string on SQL Server). For example: <pre>\$\$RAISE_EXCEPTION[MyExcepti on]</pre> raises an exception.
\$\$REMOVE_MACRO[<i>macro</i>]	Low	Removes the definition of the macro indicated by the <i>macro</i> argument.
\$\$RENAME_OBJECT	Medium	Renamed tables or other database objects. For example: <pre>\$\$RENAME_OBJECT[oldtablenam e ~ , ~ newtblname]</pre>

TABLE 21: GENERAL-PURPOSE SQL MACROS

(13 of 18)

Macros	Usage	Description
\$\$SELECT_INTO_BEGIN [<i>table_name</i>]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. For example: <pre> \$\$SELECT_INTO_BEGIN[temp_ta b] SELECT * \$\$SELECT_INTO_BODY[temp_tab] FROM Old_tab WHERE 1=0 </pre> (See “Oracle Macros,” on page 407 for more additional similar macros.)
\$\$SELECT_INTO_BODY [<i>table_name</i>]	High	Creates a table from a SELECT statement. Expands into a CREATE TABLE AS or a SELECT INTO statement. See \$\$SELECT_INTO_BEGIN .
\$\$SMALLDATE	Medium	Declares a SMALLDATETIME. See \$\$EPIINT .
\$\$SMALLINT	Medium	Declares a double-byte integer. See \$\$EPIINT .
\$\$SQLSERVER [<i>expression</i>]	High	Expands into nothing if the database engine is not SQL Server. For example: <pre> SELECT COUNT(1) FROM \$\$SQLSERVER[sysobjects]\$\$OR ACLE[tabs] </pre>
\$\$SSKEY	Low	Declares the type E.piphany uses for ss_keys . See \$\$EPIINT .

TABLE 21: GENERAL-PURPOSE SQL MACROS

(14 OF 18)

Macros	Usage	Description
<code>\$\$SUBSTRING[<i>expression</i>~,~ start~,~length]</code>	Medium	Performs a substring operation. For example: <code>\$\$SUBSTRING[name~,~1~,~8]</code>
<code>\$\$SUPERNVL</code>	Medium	Converts a null value for the first argument into the second argument. The resulting column is not nullable in the schema definition of the result set. For example: <code>SELECT \$\$SUPERNVL[col1~,~'UNKNOWN']</code>
<code>\$\$TABLE_EXISTS_CONDITION[table_name]</code>	Low	Detects if a table exists. For example: <code>SELECT COUNT(1) FROM \$\$SQLSERVER[sysobjects] \$\$OR ACLE[tabs] WHERE \$\$TABLE_EXISTS_CONDITION[ta ble_name]</code>
<code>\$\$TABLE_WITH_PREFIX[database_name~,~ table_name]</code>	Medium	Qualifies a table name with a database or user name. For example: <code>SELECT source_system_key iss FROM \$\$TABLE_WITH_PREFIX[\$\$METADBNAME~,~ source_system]</code>
<code>\$\$TINYINT</code>	Medium	Declares a single-byte integer. See <code>\$\$EPIINT</code> .

TABLE 21: GENERAL-PURPOSE SQL MACROS

(15 OF 18)

Macros	Usage	Description
\$\$TO_CHAR[<i>expression</i>]	High	<p>Converts a value to a character string. In Oracle, the value must be numeric. For example:</p> <pre>SELECT \$\$TO_CHAR[\$\$NVL[MAX(coll)]]</pre> <p>As an alternative, you can use the \$\$TO_CHAR_UNIVERSAL macro, which accepts both numeric and nonnumeric values.</p>
\$\$TO_CHAR_UNIVERSAL[<i>expression</i>]	Medium	Converts a value to a character string.
\$\$TO_DATE[<i>expression</i>]	Medium	<p>Converts a value to a database date with the following format:</p> <pre>MM/DD/YYYY HH24:MM:SS</pre> <p><i>MM</i> is the month in two-digit notation, <i>DD</i> is the two-digit day, <i>YYYY</i> is the year, <i>HH24</i> is the hour in 24-hour notation, <i>MM</i> is the two-digit minute, and <i>SS</i> is the two-digit second.</p>
\$\$TO_DATEFMT[<i>expr</i>~,~<i>format</i>]	Low	Converts expression to a date type with the appropriate Oracle format (<i>format</i> is ignored on SQL Server).
\$\$TO_EPIDATE[<i>expression</i>]	High	<p>Converts a date to the string format preferred by EpiChannel. This macro should be used for all columns that have physical type SMALLDATE. For example:</p> <pre>Select coll ss_key, \$\$TO_EPIDATE[date_col] date_modified from zork</pre>

TABLE 21: GENERAL-PURPOSE SQL MACROS

(16 OF 18)

Macros	Usage	Description
<code>\$\$TO_HHMMSS[<i>datetime_var</i>]</code>	Medium	Converts a datetime variable to a string of the form: <i>HHMMSS</i> where <i>HH</i> represents the hour, <i>MM</i> represents the minute, and <i>SS</i> represents the second.
<code>\$\$TO_INT[<i>expression</i>]</code>	Medium	Converts <i>expression</i> to an integer type.
<code>\$\$TO_NUMBER[<i>expression</i>]</code>	Medium	Converts an expression to a number. For example: <code>\$\$TO_NUMBER['123']</code> returns 123.
<code>\$\$TO_TIME[<i>expression</i>]</code>	Medium	Converts its argument to a time representation. For example: <code>SELECT \$\$TO_TIME[\$\$DBNOW]</code>
<code>\$\$TO_YYYYMMDD[<i>expression</i>]</code>	Medium	Converts a data to a YYYYMMDD string.
<code>\$\$TRANSLATE_BEGIN</code>	Medium	Searches expression for occurrences of each <i>searchval</i> and returns the <i>translationval</i> . Note that
<code>\$\$TRANSLATE_VAL[<i>expression</i>~,~ <i>searchval</i>~,~ <i>translationval</i>~,~ <i>nestedcall</i>~,~ <i>\$\$TRANSLATE_ELSE</i>[<i>otherterm</i>]]</code>		<code>\$\$TRANSLATE_VAL</code> terms should be nested inside of each other. An optional <code>\$\$TRANSLATE_ELSE</code> can be nested inside the final <code>\$\$TRANSLATE_VAL</code> . For example:
<code>\$\$TRANSLATE_ELSE</code>		
<code>\$\$TRANSLATE_END</code>		<code>\$\$TRANSLATE_BEGIN</code> <code>\$\$TRANSLATE_VAL['abcdef'~,~ bce~,~ 'Value1'~,~ \$\$TRANSLATE_VAL['abcdef'~,~ cde~,~ 'Value2'~,~ \$\$TRANSLATE_ELSE['Other']]]</code> <code>\$\$TRANSLATE_END</code> returns Value2.

TABLE 21: GENERAL-PURPOSE SQL MACROS

(17 of 18)

Macros	Usage	Description
\$\$TRANSTYPE[<i>name</i>]	High	Returns the transtype number corresponding to the name that is the first argument of the macro.
\$\$UNKNOWN_DATE	Medium	Expands to a date of the form <i>MM/DD/YYYY</i> (01/01/1990 by default). You can set a new value in this same format with EpiCenter Manager. See “Macros.” on page 241.
\$\$VAR[<i>variable_name</i>]	Medium	References a database variable. For example: <pre>SELECT 'PROCESSED', \$\$VAR[processed], 1100 \$\$NO_FROM_LIST\$\$EOS</pre>
\$\$VAR_ASSIGN_BEGIN[<i>variable_name</i>]	Medium	Assigns to a database variable. For example: <pre>\$\$VAR_ASSIGN_BEGIN[max_key] SELECT \$\$TO_CHAR[\$\$NVL[MAX(coll)-,-1]] \$\$VAR_ASSIGN_INTO[max_key] FROM table2 \$\$VAR_ASSIGN_END</pre>
\$\$VAR_ASSIGN_END	Medium	Assigns to a database variable. See \$\$VAR_ASSIGN_BEGIN .
\$\$VAR_ASSIGN_INTO[<i>variable_name</i>]	Medium	Assigns to a database variable. See \$\$VAR_ASSIGN_BEGIN .
\$\$VARCHAR_5	Medium	Declares a variable-width character datatype that holds a maximum of 5 characters. See \$\$EPIINT .
\$\$VARCHAR_15	Medium	Declares a variable-width character datatype that holds a maximum of 15 characters. See \$\$EPIINT .

TABLE 21: GENERAL-PURPOSE SQL MACROS (18 OF 18)

Macros	Usage	Description
\$\$VARCHAR_25	Medium	Declares a variable-width character datatype that holds a maximum of 25 characters. See \$\$EPIINT.
\$\$VARCHAR_50	Medium	Declares a variable-width character datatype that holds a maximum of 50 characters. See \$\$EPIINT.
\$\$VARCHAR_100	Medium	Declares a variable-width character datatype that holds a maximum of 100 characters. See \$\$EPIINT.
\$\$VARCHAR_255	Medium	Declares a variable-width character datatype that holds a maximum of 255 characters. See \$\$EPIINT.

DATABASE-SPECIFIC SQL MACROS

This section lists macros and discusses concerns that apply to specific database servers.

ORACLE MACROS

Oracle-specific SQL macros control the physical characteristics of an Oracle EpiCenter. Oracle tables are stored in a logical entity called a tablespace. Because of the various size requirements of EpiCenter objects, such as fact tables and indexes, dimension tables and indexes, EpiCenter allows you to configure which tablespace is used for each object type. When appropriate, the expansions of Oracle-specific SQL macros refer to tablespace names.

TABLE 22: ORACLE-SPECIFIC SQL MACROS

Macro	Description
\$\$ANALYZE_TABLE[table_name]	Performs a size-based analysis of the indicated table.
\$\$DIMINDEX_TABLESPACE	Used for dimension indexes (including those on aggregates and mini-dimensions).
\$\$DIMTABLESPACE	Used for dimension tables (including aggregates and mini-dimensions).
\$\$FACTINDEX_TABLESPACE	Used for fact indexes (including those on aggregates and clusters).
\$\$FACTTABLESPACE	Used for fact tables (including aggregates and clusters, as well as temporary objects used during semantics).
\$\$METATABLESPACE	Expands to the name of the tablespace to use for metadata tables on your system.
\$\$SELECT_INTO_BEGIN_OPT[table_name, option_string]	Expands to a CREATE TABLE statement. The option_string argument passes a list of options to the resulting statement.
\$\$SELECT_INTO_BEGIN_TS[table_name, tablespace]	Expands to a CREATE TABLE statement. The tablespace argument passes the name of the desired tablespace for the new table to the resulting statement.
\$\$TEMP_TABLESPACE	Used for all Application Server temporary tables (those needed for query post-processing at runtime).

Normally, the values for these macros are set to match the names of the tablespaces as they are created by the Oracle initialization script provided by E.piphany as described in the *E.piphany e.4 Installation Guide*.

To use alternate names for these tablespaces, you need to run SQL statements such as the following against your EpiMeta database.

```
update translation_Actual set actual_string =  
'your_tablespace_name' where store_type = 'Oracle' and  
translation_string = 'FACTTABLESPACE'
```

SQL SERVER DATA TYPES AND E.PIPHANY MACROS

Fact semantic types that use aggregation operators, such as SUM(), are sensitive to decimal data types such as NUMBER(*x*,*y*). When new tables are created with SELECT INTO statements based on aggregates of these data types, columns that use those data types expand to hold the largest decimal value. This expansion can unnecessarily increase the overall size of the fact table. For this reason, both FACTQTY and FACTMONEY map to the MONEY data type in SQL Server.

SQL SERVER MACROS

The following macros apply to SQL Server.

TABLE 23: SQL SERVER MACROS

Macro	Usage	Description
<code>\$\$TIMESTAMP_FILTER[column_name]</code>	High	<p>Expands to a “one-sided” SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run. This extracts “everything since last time.”</p> <p>This compares timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type, not as a time or date.</p>
<code>\$\$TIMESTAMP_LAST_VALUE</code>	High	<p>Expands to the “current timestamp” as of the start of the last run. This expands to a value, not to an expression, allowing you to make your own expressions.</p>
<code>\$\$TIMESTAMP_RANGE_FILTER[column_name]</code>	High	<p>Expands to a “two-sided” SQL comparison expression that requires the column to be greater than or equal to the current timestamp as of the start of the last run, and less than or equal to the current time as of the start of the current run. This extracts “everything since last time, but not including that data that changes while the extractions are running.”</p> <p>This compares timestamps, which actually have no time or date information in them. The column used for the comparison must be declared as a timestamp type, not as a time or date.</p>

SYSTEM-CALL MACROS

E.piphany system-call macros allow you to encapsulate operating-system commands in extraction jobs. You typically use system-call macros to pass filenames or user names that have been stored in metadata to operating-system commands. Many commands, such as **RENAME** and **DIR**, are unable to read a database, and few commands can read E.piphany-created structures.

SYSTEM-CALL MACRO SYNTAX

`$$MACRO`

`$$MACRO[argument]`

`$$MACRO[arg1, arg2]`

Names specific data stores can be abstracted within the job definition through the use of data-store roles. See “System Calls,” on page 79 for more information about roles. Unless stated otherwise, the arguments for a macro are the names of data-store roles that have been defined for an extraction job.

You can use multiple arguments in most cases, which results in an expansion of each argument. Note that the expansion of both of the following macros is the same:

`$$macro[arg1, arg2]`

`$$macro[arg1] $$macro[arg2]`

If a macro reference does not match the name of a macro that has been defined, the extraction job halts with an error. If the argument to a macro is a role name, and the job does not define that role, the extraction job halts with an error.

Table 24, on page 412 lists the E.piphany system-call macros.

TABLE 24: EPIPHANY SYSTEM-CALL MACROS

(1 OF 4)

Macros	Purpose	Usage	Description
<code>\$\$AGG</code>	Child Procs	High	The name of the AggBuilder executable in <code>\$\$EPIBIN</code> . For example: <pre> \$\$AGG \$\$EXC_ARGS -j \$\$JOB_NAME </pre>
<code>\$\$APPSERVERHOST</code>	Registry	Low	The value of this Registry variable.
<code>\$\$APPSERVERPORT</code>	Registry	Low	The value of this Registry variable.
<code>\$\$CHARTSLOGFILE</code>	Registry	Low	The value of this Registry variable.
<code>\$\$CHARTSOUTPUTDIR</code>	Registry	Low	The value of this Registry variable.
<code>\$\$DATABASE[role]</code>	Database Login	High	Translates to the name of the database or instance when the data-store role is associated with a specific database server (that is, not a data store of type ODBC or File). For example: <pre> isql /S \$\$SERVER[MyRole] /U \$\$USER[MyRole] /P \$\$PASSWORD[MyRole] /d \$\$DATABASE[MyRole] /w 300 /i /Q "gen_tests_run" </pre>
<code>\$\$DBVENDOR[role]</code>	Database Login	Medium	Translates to the vendor of the database server associated with a data-store role.
<code>\$\$DEBUG_LEVEL</code>	Command Line	Low	Translates to the current verbosity level of EpiChannel. Use this to pass EpiChannel's verbosity onto the subprocesses it spawns via system calls.

TABLE 24: EPIPHANY SYSTEM-CALL MACROS

(2 OF 4)

Macros	Purpose	Usage	Description
<code>\$\$DIRNAME[role]</code>	File ID	Medium	<p>Translates to the directory name of the data store associated with <i>role</i>, without the last filename component and without a trailing slash. If the role is WorkingDir, the directory name's last component is a unique subdirectory generated for this particular run of EpiChannel. For example:</p> <pre>echo "DIRNAME is " \$\$DIRNAME[WorkingDir]</pre>
<code>\$\$DSN</code>	Database Login	Medium	<p>Translates to the ODBC connection string for the database. This string may be generated even for databases accessed using native APIs.</p>
<code>\$\$EPIBIN</code>	Child Procs	Medium	<p>The name of the Win32 directory under the InstanceRootDir Registry variable.</p>
<code>\$\$EXC</code>	Child Procs	High	<p>The name of the extract.exe executable in <code>\$\$EPIBIN</code>.</p>
<code>\$\$EXC_ARGS</code>	Child Procs	High	<p>The recognized portions of the extract.exe command line.</p>
<code>\$\$EXC_CMD</code>	Child Procs	Medium	<p>The extract.exe program and its arguments other than job name. You can use this to fire subsidiary runs. For example:</p> <pre>\$\$EXC_CMD -j performance</pre>

TABLE 24: EPIPHANY SYSTEM-CALL MACROS

(3 OF 4)

Macros	Purpose	Usage	Description
\$\$FILENAME[<i>role</i>]	File ID	Medium	Translates to the filename of the data store associated with <i>role</i> . If the role is WorkingDir , the file name is the name of the EpiChannel log file. For example: <pre>echo "FILENAME is " \$\$FILENAME[Working Dir]</pre>
\$\$INSTANCE_NAME	Registry	Medium	The name of the instance's Registry subtree.
\$\$INSTANCEROOTDIR	Child Procs	Low	Value of the InstanceRootDir Registry variable.
\$\$JOB_NAME	Child Procs	High	The name of the current job.
\$\$PASSWORD[<i>role</i>]	Database Login	High	The password for the user of the database-server or host associated with the indicated data-store role.
\$\$PATH[<i>role</i>]	File ID	Medium	Translates to the full pathname of the file that is associated with <i>role</i> .
\$\$PROGRAM_NAME	Child Procs	Low	The name of the current extract.exe program.
\$\$REGISTRY_EPIPATH	Registry	Low	Name of the E.piphany Registry key.
\$\$SERVER[<i>role</i>]	Database Login	High	Translates to the name of the database server for the data store that is associated with <i>role</i> . For SQL Server, this is the hostname of the computer on which the database server resides. For Oracle, this is the SQLNet ID (SID). SERVER and SQLNET are identical in behavior and can be interchanged. The data-store type must be a specific database server (not ODBC or File.)
\$\$SQLNET[<i>role</i>]			

TABLE 24: E.PIPHANY SYSTEM-CALL MACROS

(4 OF 4)

Macros	Purpose	Usage	Description
\$\$USER	Database Login	High	The user name for the database-server or host associated with the indicated data-store role.
\$\$VERSION[<i>role</i>]	Database Login	Low	Translates to the version number (or string) of the database server that is associated with <i>role</i> . The data-store type must be a specific database server (not ODBC or File).

09625518.072500

09:25:18 07:25:00

EPICENTER CONFIGURATION

The Configuration dialog box shows the various configurations for your EpiCenter. To view or modify configuration settings, choose **Configuration** from the **EpiCenter** menu. The Configuration dialog box that is displayed has three tabs: General, Transaction Types, and Measure Units.

The Configuration dialog box is the user interface for the `config_master` metadata table that contains various system parameters in name/value format.

GENERAL SETTINGS

The configuration data that appears in the General tab of the Configuration dialog box has been set for a standard EpiCenter. Other than entering your e-mail password, the information should be correct, or require minimal alteration. If you need to modify these settings:

1. Select the key in the list.
2. Enter a new value in the **Value** textbox.
3. Click **Update**.

All of the values are defined for you in the dialog box. Note the following:

- **Mail Password**
The e-mail password that is used when sending extraction notices. You need to change this password after installing the E.piphany software.
- **Mail Profile Name**
The default e-mail address for the EpiCenter Manager user.

- **date_type**

The type of calendar that is used. The default value is **Calendar**.

date_445 represents the 13-week-per-quarter calendar in which the months in the quarter are defined as consisting of 4 weeks, 4 weeks, and 5 weeks.

- **start_year**

The earliest year for which data can be extracted into the EpiMart. The default is 1990.

- **end_year**

The ending year of the EpiMart. The date range of the EpiMart should be at least as large as any dates that are found in the data that that you intend to extract.

For users to get the best results when forecasting trends, build the date dimension as far into the future as you plan to forecast. (Currently, the maximum prediction is three years past the last date that has recorded data.) If the date dimension is not built out far enough, end users receive columns on their reports with names such as:

Dec 1999, Second Next, Third Next,

instead of:

Dec 1999, Dec 2000, Dec 2001.

- **fiscal_year_start_month**

The starting month of the fiscal year. If the chosen month is other than January, then the actual starting year is one less than the **start_year** value (thus, January 1 of the calendar **start_year** is included in the earliest available fiscal year).

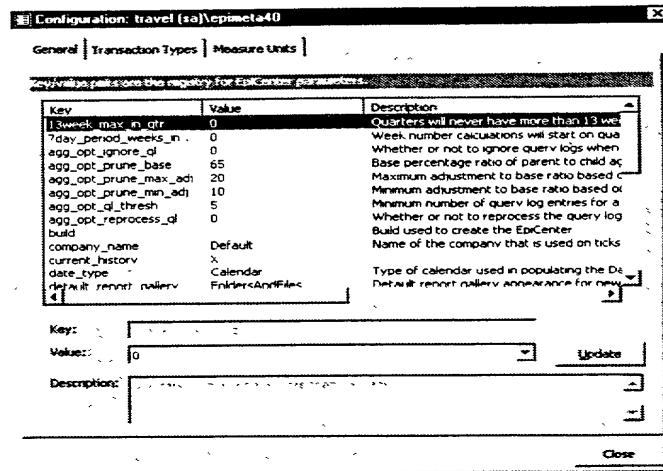
- **start_day_445**

The beginning day of the 445 quarter.

09625518.072500

- **version**
The version number of this EpiCenter Manager.
- **week_start_day**
The day of the week that is the first day of the week. Sunday is the default.

FIGURE 113: CONFIGURATION DIALOG BOX: GENERAL TAB



TRANSACTION TYPES

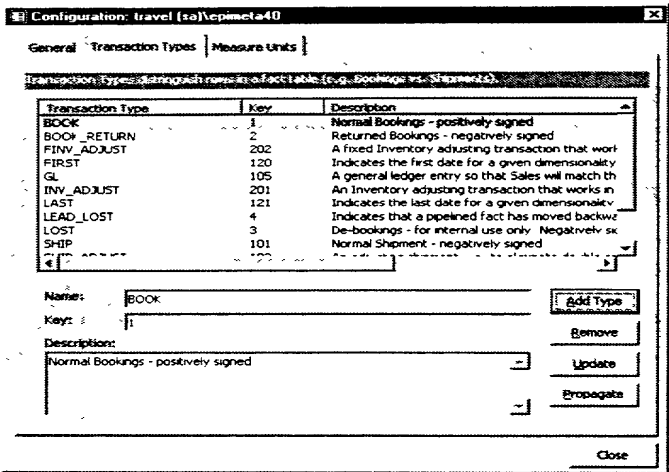
The current transaction types are shown in the Transaction Types tab (Figure 114) of your Configuration dialog box. These are the typical transaction types for a generic installation. Your site may need additional transaction types for complicated measures.

Use this tab to customize the transaction types for your site. Keys 1-99 are reserved for booking transaction types. Keys 101-199 are reserved for shipping transaction types. Keys 10,000-20,000 are reserved for use by E.piphany.

After modifying the transaction types, click **Update**, which writes the new data to the EpiMeta. Renaming existing transaction types affects the import of metadata. Instead of renaming a transaction type, create a new one that is identical except for the name, and then delete the old one. For more information, see “Transaction Types,” on page 493.

Use the **Propagate** button to synchronize the transtype_0 table in the EpiMeta and EpiMart databases.

FIGURE 114: CONFIGURATION DIALOG BOX: TRANSACTION TYPES TAB



MEASURE UNITS

Measure units define how currency units, percentages, and numerical units are displayed by default on Web Pages. To define a measure unit, enter its name and symbol, such as the dollar or per cent sign. The symbol character can be at most one character. To input the Yen sign and other similar characters, use the Windows character map to copy and paste the symbol.

For a currency unit, if this EpiCenter uses currencies from different countries, check the **Multi-currency** option. Multi-currency controls how the system handles currencies. (A GROUP BY is forced on the currency dimension so that the system does not calculate different currencies, such as French francs and British pounds, as the same currency.) Also, check **Postfix** if the symbol should follow the number: 100% is postfix, whereas \$100 (the default) is prefix.

Add any description for your reference in the Description textbox, and click **Add Unit**. The system generates a hidden key for the unit. EpiCenter Manager uses this key when flagging a measure with a unit designation.

FIGURE 115: CONFIGURATION DIALOG BOX: MEASURE UNITS TAB

Configuration: travel (sa)\epimeta40

General | Transaction Types | **Measure Units**

Measure Units control the display of most measures.

Name	Symbol	Multi-Currency	Postfix S...	Description
CURRENCY	\$	0	0	Default cur
CURRENCY_LOCAL		1	0	Special Mul
CURRENCY_US	\$	0	0	Default cur
PERCENT	%	0	1	Uses the %
UNITS		0	0	Shows num

Name:

Symbol: ☐ Multi-currency ☐ Postfix

Description:

09625518, 072500

DATE DIMENSION FIELDS

Table 25 describes the date dimension fields that the E.piphany system uses.

TABLE 25: DATE DIMENSION FIELDS

(1 OF 4)

dim_col_name	Description
cq_and_cy_name	Calendar quarter and year name. For example, Q1 1999.
cq_name	Calendar quarter name. For example, Q1.
cy_name	Calendar year name. For example, 1999.
date_key	Primary key—date as a native date type.
day_cq_begin	Whether or not this is a day on which a calendar quarter begins. (1/0).
day_cq_end	Whether or not this is a day on which a calendar quarter ends. (1/0).
day_cy_begin	Whether or not this is a day on which a calendar year begins. (1/0).
day_cy_end	Whether or not this is a day on which a calendar year ends. (1/0).
day_fq_begin	Whether or not this is a day on which a fiscal quarter begins. (1/0).
day_fq_end	Whether or not this is a day on which a fiscal quarter ends. (1/0).
day_fy_begin	Whether or not this is a day on which a fiscal year begins. (1/0).

005270" 87552960

TABLE 25: DATE DIMENSION FIELDS

dim_col_name	Description
day_fy_end	Whether or not this is a day on which a fiscal year ends. (1/0).
day_month_begin	Whether or not this is a day on which a month or period begins. (1/0).
day_month_end	Whether or not this is a day on which a month or period ends. (1/0).
day_name	The day as a native date type.
day_name_char	Date (as a string). For example, Apr 02 1995.
day_name_char_weekday	Date (as a string) with weekday prefix. For example, Sun Apr 02 1995.
day_number_in_cq	The number of this day in the calendar quarter, starting at 1.
day_number_in_cy	The number of this day in the calendar year, starting at 1.
day_number_in_fq	The number of this day in the fiscal quarter, starting at 1.
day_number_in_fy	The number of this day in the fiscal year, starting at 1.
day_number_in_month	The number of this day in the month or period, starting at 1.
day_number_in_week	The number of this day in the week, starting at 1.
day_number_til_end_cq	The number of days until the end of the calendar quarter, ending with 1.
day_number_til_end_fq	The number of days until the end of the fiscal quarter, ending with 1.
fq_and_fy_name	Fiscal quarter and year name. For example, Q1 1999.
fq_name	Fiscal quarter name. For example, Q1.
fy_name	Fiscal year name. For example, 1999.
month_and_cy_name	The name of the month abbreviated and the calendar year. For example, Apr 1999.

TABLE 25: DATE DIMENSION FIELDS

(3 OF 4)

dim_col_name	Description
month_and_fy_name	Month (abbreviated) or period name and fiscal year. For example, Apr 1999.
month_name	The three-letter abbreviations of the month (without a year or any other value). For example, Apr.
month_number	Month or period number since the first date in the system, starting at 1.
month_number_in_cq	Month number since the start of the calendar quarter, starting at 1.
month_number_in_cy	Month number since the start of the calendar year, starting at 1.
month_number_in_fq	Month or period number since the start of the fiscal quarter, starting at 1.
month_number_in_fy	Month or period number since the start of the fiscal year, starting at 1.
month_number_til_end_cy	Number of months or periods until the end of the calendar year, ending with 1.
month_number_til_end_fy	Number of months or periods until the end of the fiscal year, ending with 1.
vacation_day	Whether or not this day is a vacation, (1/0).
week_friday	Date (as a string) for the Friday of the current week. For example, Apr 01 1994.
week_monday	Date (as a string) for the Monday of the current week. For example, Apr 01 1996.
week_number	Week number since the first date in the system, starting at 1.
week_number_cq	Week number since the start of the calendar quarter, starting at 1.

TABLE 25: DATE DIMENSION FIELDS

(4 OF 4)

dim_col_name	Description
week_number_cy	Week number since the start of the calendar year, starting at 1.
week_number_fq	Week number since the start of the fiscal quarter, starting at 1.
week_number_fy	Week number since the start of the fiscal year, starting at 1.
week_number_til_end_cq	Week number until the end of the calendar quarter, ending with 1.
week_number_til_end_cy	Number of weeks until the end of the calendar year, ending with 1.
week_number_til_end_fq	Week number until the end of the fiscal quarter, ending with 1.
week_number_til_end_fy	Number of weeks until the end of the fiscal year, ending with 1.
week_saturday	Date (as a string) for the Saturday of the current week. For example, Apr 01 1995.
week_sunday	Date (as a string) for the Sunday of the current week. For example, Apr 02 1995.
weekday	Weekday prefix; for example, Sun.
month_name	The three-letter abbreviations of the month (without any year or any other value).

PHYSICAL TYPE VALUES

The tables in this appendix define the database type translations for the physical types you select in EpiCenter Manager's Base Dimension and External Tables dialog boxes. The physical type you select is replaced by its associated database type. The translation of physical type to database type depends on your RDBMS platform.

Notes to sites that use their own database types and SQL macros:

The various fact semantic types (such as Transactional/Statelike) use SUM () operators over the fact columns. New tables are created via SELECT INTO from the result.

With decimal data types such as NUMBER (16 , 4) that logically map to a NUMERIC (16 , 4) SQL Server datatype, repeated SUM operations within SQL Server may cause an automatic increase in the column precision, ultimately resulting in a NUMERIC (38) field, which occupies 17 bytes per record. For this reason, both FACTMONEY and FACTQTY map to the MONEY database type in SQL Server. This database type occupies only 8 bytes per record

Table 26 lists SQL Server physical type values.

TABLE 26: SQL SERVER PHYSICAL TYPE VALUES

(1 of 2)

Physical Type	Database Type
BIGDATE	DATETIME
CHAR_1	CHAR(1)
DOUBLESTRING	VARCHAR(21)
EPIINT	INT

TABLE 26: SQL SERVER PHYSICAL TYPE VALUES

(2 OF 2)

Physical Type	Database Type
EPIKEY	INT
FACTMONEY	MONEY
FACTQTY	MONEY
IDENTITY	INT IDENTITY
LONGSTRING	VARCHAR(11)
MONEYSTRING	VARCHAR(21)
NUMBER(9,5)	DECIMAL(9,5)
SMALLDATE	SMALLDATETIME
SMALLINT	SMALLINT
SSKEY	VARCHAR(50)
TINYINT	TINYINT
VARCHAR_5	VARCHAR(5)
VARCHAR_15	VARCHAR(15)
VARCHAR_25	VARCHAR(25)
VARCHAR_50	VARCHAR(50)
VARCHAR_100	VARCHAR(100)
VARCHAR_255	VARCHAR(255)

Table 27 lists Oracle physical type values.

TABLE 27: ORACLE PHYSICAL TYPE VALUES

Physical Type	Database Type
BIGDATE	DATE
CHAR_1	CHAR(1)
DOUBLESTRING	VARCHAR2(21)
EPIINT	NUMBER(10)
EPIKEY	NUMBER(10)
FACTMONEY	NUMBER(19,4)
FACTQTY	NUMBER(19,4)
IDENTITY	NUMBER(10)
LONGSTRING	VARCHAR2(11)
MONEYSTRING	VARCHAR2(21)
NUMBER(9,5)	NUMBER(9,5)
SMALLDATE	DATE
SMALLINT	SMALLINT
SSKEY	VARCHAR2(50)
TINYINT	SMALLINT
VARCHAR_5	VARCHAR2(5)
VARCHAR_15	VARCHAR2(15)
VARCHAR_25	VARCHAR2(25)
VARCHAR_50	VARCHAR2(50)
VARCHAR_100	VARCHAR2(100)
VARCHAR_255	VARCHAR2(255)

Oracle Database 11g Release 1 (11.1.0.7) - New Features

09625518, 072500

WRITING STAGING SQL STATEMENTS

The basic E.piphany extraction process consists of the following phases:

1. Loading data from source systems into the E.piphany staging tables
2. Running semantic instances against these staging tables.

This appendix describes the recommended practices for writing SQL statements that populate the dimension and fact staging tables.

BASE DIMENSION STAGING SQL STATEMENTS

Base dimensions describe the physical dimension tables in EpiMart. You can use EpiCenter Manager to configure the dimension columns for each base dimension table. During an extraction job, one or more SQL statements can be executed against the base dimension staging table to populate these columns.

The purpose of the base dimension staging query is to extract the latest values from the source system. You need not be concerned with determining when a value has changed because E.piphany's semantic templates perform this task.

You can use the Template feature in the SQL Statement dialog box to provide an SQL template for a given dimension table. For example, assume you define a base dimension called Customer with the following dimension columns:

- FullName
- Age
- City
- State
- ZipCode

E.piphany Confidential

In the SQL Statement dialog box, select the option **Populates dimension table**. Choose the dimension table (Customer) from the drop-down list. Clicking the Template button will display the following SQL in the dialog box:

```
SELECT
    <YOUR EXPRESSION>                                customer_sskey,
    $$TO_EPIDATE[ $$DBNOW ]                            date_modified,
    $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]FullName,
    $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]Age,
    $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]City,
    $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]State,
    $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]ZipCode
FROM
    <YOUR TABLE>
```

Note: *The displayed template code also has a comment for each column that indicates the physical type of that column.*

This is a regular SQL statement for which you must provide the values <YOUR EXPRESSION> and <YOUR TABLE>. You must assign each column in the SELECT list a valid SQL expression for the value of its destination column. A FROM clause is required to make this a valid statement; a WHERE clause is optional.

An important point about these expressions is that null values are *not* allowed in any field of the staging table. The reason for this is that the E.piphany system uses GROUP BY statements at end-user query time to form the tables and charts of front-end applications. However, fact rows that aggregate on columns with null values are left out of the resulting reports because nulls are removed from GROUP BYs. Rather than incurring the query-time penalty for this check, EpiMart insists on non-null dimension column values.

To circumvent this problem, substitute the string 'UNKNOWN' for any null values using the \$\$NVL macro, as shown in the template. The E.piphany system will automatically generate an 'UNKNOWN' row in your dimension table. The 'UNKNOWN' value is configurable; if 'UNKNOWN' is a valid value in your dimension table, use another value.

The first two columns, **customer_sskey** and **date_modified**, were not specified in the definition of the Customer base dimension. These columns are implicitly added to the base dimension table by E.piphany's Adaptive Schema Generator and must be populated by the staging SQL. The first column is a source system key (**sskey**) and should be unique for every row in the staging table. The concept of **sskey** is important in EpiMart because it tells the semantic templates whenever a row in the staging table represents the same source system entity as a row that already exists in the dimension table. The **sskey** is a variable length string, normally of maximum length 50. It corresponds to the primary key of the source system table or the tables that make up this query.

Note: If the sskey column is of interest to end users for querying, then a dimension column populated with the same values will need to be created because the sskey is not available for Web Page configuration.

The other implicit column, **date_modified**, has the same name in all base dimension staging tables and is used to identify when a base dimension row is inserted into the EpiMart. If the source system contains a *creation date* field, then this field should be used. Otherwise, you can use the source system's expression for "right now," which causes newly extracted rows to assume the date when they were extracted into EpiMart; for Microsoft SQL Server this expression is `GetDate()`. The `$$DBNOW` macro is automatically expanded to the appropriate expression for the current date. For best results, dates should be returned as strings, for example, 5/26/1999. This can be done using the `$$TO_EPIDATE` macro, as shown in the template.

The remainder of the columns in the `SELECT` list must be populated with an appropriate expression for the meaning of that column. Any SQL expression that can be executed against the source RDBMS is valid. Also, E.piphany provides a set of SQL macros that will be automatically expanded to the correct syntax for your source system. Use of macros facilitates the cross-platform usage of your SQL statements. See Appendix A, "E.piphany Macros," for more information.

In dimension tables, source system entries with large numbers of possible values are often classified into categories or ranges. For example, a source system may record the exact ages of individuals, while a dimension might simply record the age range into which an individual falls. This kind of categorization is called *binning*, and E.piphany's `$$CBIN_*` and `$$NBIN_*` macros are designed to simplify the process.

You can also use the drop-down menu next to the template button to insert a template for a single dimension column.

DUPLICATE SSKEYS

If during a single extraction, a staging table is loaded with two or more rows with the same **sskey**, then the last row entered is used. See Appendix F, “Semantic Types,” for a description of the dimension semantic types.

DIMENSION STAGING QUERIES WITH JOINS

The E.piphany system allows the use of joins in base dimension staging queries. Star schemas typically de-normalize data structures in transactional systems into flat hierarchies, and you must be aware of what the granularity of a base dimension represents in this circumstance.

For example, you will rarely want to use a Cartesian product of two tables in a base dimension staging query, unless the **sskey** of the result set will combine the primary keys of the two tables that are being crossed. It is more common for a single table to “drive” the result set, with other tables joined through unique key lookups to provide additional textual values. For instance, a Product Master table in the source system might represent the driving table of a Product base dimension (with the **sskey** taken from the primary key of the Product Master table), but other tables with textual values for Product Line or Platform may be joined with this master table. In this case, you should ensure that the joined columns of the lookup tables are properly indexed (usually with UNIQUE indexes).

BASE DIMENSION QUERIES WITH DISTINCT FACT VALUES

Sometimes dimensions are created for which no corresponding master table exists in the source system. For instance, an Order fact may have an Order type associated with it (with several possible choices). These values are embedded directly in the rows of the source system's Order table, but no lookup table exists with all the choices. In this case, a `SELECT DISTINCT` query against the Order type column of the source system's Order table might be appropriate for populating base dimension staging tables in EpiMart. The alternative to this method is the use of degenerate dimensions in the fact table, although degenerate dimensions cannot be aggregated.

SEED DIMENSION EXTRACTION

If you are using seed dimensions in campaigns, you also need to populate the **IndivSeed** and **GroupSeed** dimensions. These dimensions have the same columns as the dimensions that correspond to the **indiv** and **group** dimension roles, respectively. You should ensure that your seed dimensions have more rows than the maximum number of cells that a user will have in a campaign, since each cell will be assigned a distinct element of the seed dimension.

CAMPAIGN AND CELL EXTRACTION

If you are using the Campaign manager, you need to extract data for the **Campaign** and **Cell** dimensions from the backfeed tables. Epiphany provides default extractors for this purpose, but you will need to modify these extractors if you have added additional columns to the **Campaign** or **Cell** dimensions. The default extractor for the Campaign dimension has the following SQL code:

```
SELECT
    $$NVL[ campaign_id ~,~ 'UNKNOWN' ]      campaign_skey,
    $$TO_EPIDATE[ rundatetime ]              date_modified,
    $$NVL[ campaign_name ~,~ 'UNKNOWN' ]     campaign_name,
    $$NVL[ campaign_label ~,~ 'UNKNOWN' ]     campaign_label,
    $$NVL[ fixed_setup_cost ~,~ ' ' ]         fixed_setup_cost,
    $$NVL[ def_rev_per_response ~,~ ' ' ]     def_rev_per_response,
    $$NVL[ def_profit_per_response ~,~ ' ' ]  def_profit_per_response,
    $$NVL[ def_cost_per_treatment ~,~ ' ' ]   def_cost_per_treatment,
    $$NVL[ def_response_rate ~,~ ' ' ]        def_response_rate,
    $$NVL[ campaign_id ~,~ 'UNKNOWN' ]        campaign_id,
    $$NVL[ campaign_code ~,~ 'UNKNOWN' ]      campaign_code,
FROM
    Campaign_BF$$CURR
```

Ordinarily, these extractors will simply copy values from backfeed-table columns to the dimension columns with the same names. As usual, the `$$NVL` macro should be used to ensure that no null values are copied. The backfeed table has a column called `rundatetime`, which records the time that the campaign was run, that should be used for the date. The value of **campaign_id** is generally also used for the source system key. You may want to use the `$$NBIN_*` and `$$CBIN_*` macros to divide numerical values (such as costs) into ranges. The backfeed tables for the Campaign table are named **Campaign_BF_A** and **Campaign_BF_B**, so data should be extracted from `Campaign_BF$$CURR`.

The same considerations apply to the extractor for the Cell table.

The built-in columns of the Campaign and Cell dimensions are described on page 443.

Note: While Campaign and Cell dimension data is ordinarily extracted from the backfeed tables, these dimensions can also be populated from external data sources.

FACT STAGING SQL STATEMENTS

SQL statements that populate fact staging tables are generally more complex than the ones used to load dimension staging tables. As with base dimension tables, the columns of the SELECT statements are determined by the metadata definition of the fact table in addition to certain implicit rules.

To illustrate this point, assume that you define an Order fact with the following dimension roles:

- CustomerBillTo
- Product
- CustomerShipTo
- SalesPerson

The EpiCenter contains a single degenerate dimension called OrderNumber, and the Order table has two fact columns: **net_price** and **number_units** that represent the extended amount for an order line item, along with the quantity.

Clicking the Template button on the SQL Statement dialog box (with the **Populates fact table** option selected and the Order table selected in the drop-down list) displays the following SQL in the dialog box:

```

SELECT
  <YOUR EXPRESSION>                                ss_key,
  $$TO_EPIDATE[ <YOUR EXPRESSION> ]                date_key,
  <YOUR EXPRESSION>                                transtype_key,
  <YOUR EXPRESSION>                                process_key,
  $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]customerbillto_sskey,
  $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]product_sskey,
  $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]customershipto_sskey,
  $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]salesperson_sskey,
  $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]ordernumber_key,
  $$NVL[ <YOUR EXPRESSION> --,-- 0 ]                net_price,
  $$NVL[ <YOUR EXPRESSION> --,-- 0 ]                number_units
FROM
  <YOUR TABLE>

```

As with base dimension staging queries, you must identify what a row in this fact table represents. Based on the columns in this example, a row seems to indicate a line item of a sales order. (In this case, the assumption is that the salesperson gets full credit for a line item; another interpretation of this fact row might be a particular amount of credit that a salesperson received for an order line item.) Typically, the FROM clause of this query would join the Order Line Item table to the Order Header table in the source system.

The columns in the SELECT list can now be divided into these categories:

- Implicit columns that were added automatically
- Dimension role foreign keys
- Degenerate dimension keys
- Fact numeric columns

09625518.072500

First, consider implicit columns. As with base dimensions, each fact staging row contains an **ss_key** (notice the difference in spelling) that uniquely identifies this row in the source system. In this example, the **ss_key** might be a concatenation of the Order Number with the Order Line Number (since this combination is presumably unique). **ss_keys** will be used on subsequent extractions to prevent duplicate copies of the fact row from being created in EpiMart.

The **date_key** indicates when the fact occurred. Since time is a central component of EpiMart, each fact table must contain this column. Many facts are time based; in this example, **date_key** represents the time when the order was placed. However, if time is not important for this fact, then the current system time can be used as a placeholder. Note that **date_key** is granular only to that single *day* when the fact occurred. For best results, the fact SQL Statement should return the day as a string, for instance, 5/1/1999. E.piphany recommends that you use the `$$TO_EPIDATE` macro to ensure that dates are in the proper format.

Transaction type is another central concept of fact table processing in EpiMart. The SQL statement should return a numeric key that matches with one of the transaction types defined on the Configuration dialog box in EpiCenter Manager. For example, the Order fact might hold both Bookings and Shippings, and **transtype_key** would identify which fact staging rows were which. (See Appendix F, "Semantic Types," for more information about **transtype_key**.)

Note: Transaction type values in the range 10000-20000 are reserved for use by E.piphany.

A fact staging table can contain different types of rows that need to be handled in different ways by the semantics. The **process_key** identifies rows from the fact table to be processed in a specific way by a semantic instance. A value of 1 indicates a transactional fact, and a value of 2 indicates a state-like fact.

Next, you must enter values for each of the dimension role foreign keys. Notice that the names of the columns in the SQL template are **DimRoleName_sskey**. These fields refer back to **sskeys** of the base dimension tables for this fact. You need to understand the meaning of each base dimension table to ensure that the keys resolve properly. If the **sskey** of the Product base dimension is taken from a Product Master list in the source system, then **product_sskey** in the Order table must also refer to an entry in the Product Master. If a base dimension is the cross-product of two source system tables, the fact staging keys for that dimension must also represent a unique cross-product entry.

*Note: If you have defined additional dimension roles that refer to the date dimension, then the names of the foreign keys for those roles end in key rather than sskey. For example, if you have defined a dimension role called **inquiry_date** that refers to the date base dimension, then the name of the foreign key for that dimension role is **inquiry_date_key**.*

Degenerate keys in the fact staging query should be populated with string values. In the example above, the **ordernumber_key** field would probably be populated with the actual primary key of the Order Header table, such as Order Number 253AD56.

Finally, the numeric columns represent the actual quantities and raw amounts that are associated with each fact entry. Each column should be an additive amount for correct front-end query results. For instance, total dollar amounts for a line item should be populated instead of unit prices because unit prices cannot be added across fact rows.

IND_GROUP_JOINER EXTRACTION

If you are using the List Manager or Campaign Manager, you need to extract data for the **Ind_Group_Joiner** fact table. This table specifies the association between members of the dimensions that have the **indiv** and **group** dimension roles. The usual format for **Ind_Group_Joiner** extraction is the following:

```
SELECT
  <YOUR EXPRESSION>                                ss_key,
  $$TO_EPIDATE[ $$DBNOW ]                          date_key,
  1                                                  transtype_key,
  2                                                  process_key,
  $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]group_sskey,
  $$NVL[ <YOUR EXPRESSION> --,-- 'UNKNOWN' ]indiv_sskey,
  $$NVL[ <YOUR EXPRESSION> --,-- 0 ]'              member
FROM
  <YOUR TABLE>
```

Each member of the dimension with the **indiv** role must be associated with exactly one member of the dimension with the **group** role. Note the following:

- The **member** column indicates that the individual with **indiv_sskey** belongs to the group with **group_sskey**. The **member** column can have any non-zero value. In most cases, the value 1 is used.
- The value of **process_key** should be 2. Note that this implies that new data should be merged into the **Ind_group_joiner** fact table using a Transactional/State-Like semantic type (see “Transactional/State-like,” on page 455).
- Generally, the value of **ss_key** should be the same as the value of **indiv_sskey**. Making these values the same ensures that the Transactional/State-Like semantic type properly updates the table when an individual changes group membership.
- Ordinarily, the only **transtype** value that is used in the **Ind_group_joiner** table is 1.

SEEDJOINER EXTRACTION

If you are using seed dimensions in campaigns, you need to populate the **SeedJoiner** fact table. The **SeedJoiner** fact table has the same role for the **IndivSeed** and **GroupSeed** dimensions that the **Ind_Group_Joiner** has for the individual and group dimensions. Therefore, extraction for the **SeedJoiner** is completely analogous to extraction for **Ind_Group_Joiner**. A row of the **SeedJoiner** table specifies that the member of **IndivSeed** with **sskey** value **indiv_sskey** belongs to the member of **GroupSeed** with **sskey** value **group_sskey**.

Note that only seed dimension members that have SeedJoiner fact table entries are available for seeding. Therefore, you should ensure that you have a SeedJoiner fact for every member of your seed dimensions.

COMMUNICATION EXTRACTION

If you are using the Campaign manager, you also need to extract data for the **Communication** fact table from the backfeed table. E.piphany provides a default extractor for this purpose. The default extractor for the Communication table has the following SQL code:

```
SELECT
    ($$TO_YYYYMMDD[rundatetime] $$CAT $$TO_HHMMSS[rundatetime] $$CAT
        group_sskey $$CAT indiv_sskey $$CAT cell_sskey) ss_key,,
    $$TO_EPIDATE[ rundatetime ]                date_key,
    $$TRANSTYPE[ BACKFEED]                      transtype_key,
    1                                           process_key,
    group_sskey,
    indiv_sskey,
    campaign_sskey,
    cell_sskey,
    treatment_code_key,
    1                                           occur
FROM
    Communication_BF$$CURR
```

As with the extractors for the Campaign and Cell dimensions, most columns are simply copied from the backfeed table. Note the following:

- The communication backfeed table contains a column called **rundatetime** that records the time at which the campaign was run.
- The default **transtype** value for a communication, which indicates that a treatment has been applied, is entered using the `$$TRANSTYPE[BACKFEED]` macro. Additional values may be defined for other campaign-related facts.
- The **occur** column indicates that the fact has occurred. It should be given a value of 1.

USING EXTERNAL TABLES AS INPUTS TO STAGING QUERIES

Sometimes it may be necessary to bring data into EpiMart external (temporary) tables before performing any joins; for example, if the source system's SQL limits your ability to manipulate the data. The full power of EpiMart's RDBMS engine can then be used to load the staging tables. In this case, the following sequence of actions is usually employed:

1. Drop any indexes on the external tables for fast loading.
2. Load the external tables from the source system.
3. Create any indexes on external tables needed for fast joins.
4. Load the staging tables using queries against the EpiMart External tables. All query plans should use the indexes built in Step 3.

BUILT-IN COLUMNS IN CAMPAIGN AND CELL DIMENSIONS

The Campaign and Cell dimensions have several built-in columns that cannot be removed or modified. These columns are found in both the dimension tables and the backfeed tables.

The Campaign dimension has the following built-in columns:

- **campaign_code**: A unique identifying code for the campaign
In the backfeed table, this code generally has a value that has been assigned by the end user. If the end user does not assign this code when saving a campaign, then a unique string is automatically generated the first time the campaign is saved. This is the backfeed table field that is used to identify a campaign for exclusion purposes.
- **campaign_id**: An identifying code for the campaign version
In the backfeed table, this value is automatically incremented every time a campaign is saved.
- **campaign_label**: A descriptive identifier for the campaign
- **def_cost_per_treatment**: The default cost per treatment in the campaign
- **def_profit_per_response**: The default profit per response in the campaign
- **def_response_rate**: The default response rate for the campaign
- **def_rev_per_response**: The default revenue per response in the campaign
- **fixed_setup_cost**: The fixed setup cost of the campaign

The Cell dimension has the following built-in columns:

- **treatment_code**: A unique identifying code for the cell
In the backfeed table, this code generally has a value that has been assigned by the end user. If the end user does not assign this code when saving a campaign, then the value is set to 'N/A'. This is the backfeed table field that is used to identify a cell for exclusion purposes.
- **campaign_id**: The version code for the associated campaign
- **cell_id**: A unique identifier for the cell and campaign version
In the backfeed table, this is formed by concatenating **campaign_id** with the cell node number. Within a campaign, this code is unique.
- **cell_label**: A descriptive identifier for a cell
- **cell_position**: The position of the cell in the segmentation tree
- **cell_size**: The number of individuals or groups in the cell

- **est_cell_cost**: The estimated cost of the cell
- **est_cell_profit**: The estimated profit for the cell
- **est_cell_response_rate**: The estimated response rate for the cell
In the backfeed table, if this value has not been specified by the end user, then the **def_response_rate** value of the associated campaign is used.
- **est_cell_revenue**: The estimated revenue rate for the cell
- **est_profit_per_response**: The estimated profit per response for the cell
In the backfeed table, if this value has not been specified by the end user, then the **def_profit_per_response** value of the associated campaign is used.
- **est_revenue_per_response**: The estimated revenue per response for the cell
In the backfeed table, if this value has not been specified by the end user, then the **def_rev_per_response** value of the associated campaign is used.
- **output_channel_label**: The label of the output channel associated with the cell
- **output_format_label**: The label of the output format associated with the cell
- **output_processor_label**: The label of the output processor associated with the cell
- **treatment_unit_cost**: The cost of a single treatment in the cell
In the backfeed table, if this value has not been specified by the end user, then the **def_cost_per_treatment** value of the associated campaign is used.

09/25/18 07:25:00

SEMANTIC TYPES

This appendix describes the dimension and fact semantic types.

DIMENSION SEMANTIC TYPES

The dimension semantic types are Slowly Changing Dimensions, Latest Dimension Value, First Dimension Value, Initial Load Dimension, and Rewrite Dimension.

SLOWLY CHANGING DIMENSIONS

A Slowly Changing Dimension is a dimension in which the attributes or hierarchy of the dimension can change over time, but historical data is not restated. Two examples follow:

- A national sporting goods chain has stores divided into three regions. On January 1, 1998, the chain reorganizes its regions, moving Denver from the Central to the Western region. Their 1998 sales forecasts take into account that the Denver store is in the Western region, but they do not want to recalculate forecasts and actuals from previous years. Using the Slowly Changing Dimensions semantic type, sales for Denver can be aggregated up to the Central region through 1997. Beginning January 1, 1998, Denver sales are applied to the Western Region.

- On May 7, 1999, one of this store's customers changes careers and doubles her income. Since the store analyzes customer purchases based on demographics, all of this customer's new purchases should be analyzed with her new demographic data. However, all of her previous purchases should still be analyzed with her old demographic data. Using the slowly changing dimensions semantic type, all of this customer's purchases are recorded based on her demographics at the time of the purchase.

The Slowly Changing Dimensions semantic type accomplishes the following logic:

- Rows with the same **sskey** in the staging table will be eliminated following in a "last in wins" rule. The last row is determined by the special column called **ikey**, which is created by EpiChannel and is automatically incremented during normal extractions. The highest **ikey** row for a given **sskey** will be accepted.
- New rows are created by searching through the dimension-staging table for new **sskey** values, or **sskey** values that have one or more dimension column changes from the last known values. Each of these cases creates a new row in the dimension table. The mapping row for that **sskey** points to the latest dimension row with that **sskey** value.
- In the EpiChannel log file, new **sskey** rows are reported in the **Inserted** column. The number of changed rows is reported in the **Modified** column. The number of rows in the staging table is reported in the **Processed** column.
- The key column of the dimension table becomes the primary-key index. Each dimension column is also Indexed (non-uniquely). The mapping table is indexed (primary key) on **iss**, **sskey**. You can use the Dimension Column dialog box in EpiCenter Manager to disable the indexing of individual columns.

iss is used when two source systems have the same **sskey** values (such as when the SAP and Vantive systems both have a Customer #2 record). **iss** is determined by the source-system identifier that is selected in the General tab of the Data Store dialog box.

- The column *dimension_key_REAL* (where *dimension* is the dimension column name) is set to the first key value for each **sskey**. In other words, when a new **sskey** is discovered, the **REAL** key is set to the new dimension key value. Subsequent dimension rows for this **sskey** will retain the original **REAL** key value.

Note the following considerations:

- Do not allow dimension column values to *oscillate* unpredictably. (See “First Dimension Value,” on page 451 for more information.) In particular, do not rely on **key** filtering of duplicate **sskey** values if two or more rows during a single extraction might have different values for one or more dimension columns. The reason is that a new row will be created in the dimension table for every extraction for which a change is recorded. Consequently, two values can “compete” with each other, forcing an unending sequence of row creation in the dimension table.
- Rows can be removed from dimension tables after they have been extracted with an explicit delete or truncation, or through use of the Initial Load Dimension semantic.
- The **UNKNOWN** dimension row always has a key value of 1. Since the **key** value is constant, the **UNKNOWN** row does not have an entry in the mapping table.
- Slowly Changing Dimension cannot be used in Initial jobs.

LATEST DIMENSION VALUE

The Latest Dimension Value updates rows instead of performing Slowly Changing Dimensions. It applies changes retroactively to a dimension. Thus, the changes take effect for all historical data, as well as for current and future loads.

For example, assume that a sporting goods store has a category historically called **Rollerblades**. Now that they are selling other brands, the store wants to change the category to **In-line Skates**. By using the Latest Dimension Value semantic type, this change can affect all of the historical data because all previous sales of **Rollerblades** are now labeled **In-line Skates**. Consequently, the store can compare year-to-year sales of all in-line skates.

This semantic type has the same duplicate **sskey** filtering as Slowly Changing Dimensions. Use this semantic type for an implementation that “restates history” when a source dimension table changes.

Note the following considerations:

- New **sskeys** are inserted in both the dimension table and mapping table. Existing **sskeys** with one or more changed dimension columns are updated in place in the dimension table (that is, the same dimension row is used) with the latest values.
- In the EpiChannel log file, new **sskey** rows are reported in the **Inserted** column. The number of changed rows is reported in the **Modified** column. The number of rows in the staging table is reported in the **Processed** column.
- Latest Dimension Value has the same indexing as Slowly Changing Dimensions.
- The **REAL** column is always equal to the dimension key.
- Latest Dimension Value has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.
- Latest Dimension Value cannot be used in Incremental or Initial jobs.
- All fact aggregate tables must be rebuilt after the Latest Dimension Value semantic is used.

FIRST DIMENSION VALUE

The First Dimension Value semantic type ignores any changes to a dimension. Values present when the row was first inserted are preserved forever, regardless of any future changes. This semantic never modifies a row after it has been seen.

You might want to use this type if your source data comes from two systems that are not in complete agreement with each other. For instance, one system might have Customer #12345 as David Anderson, and another system might have the same customer as David Andersen. Ideally, you would determine which one was in error and correct it.

In the meantime, you could choose to apply the first value read and to ignore the other. (This is a good method for avoiding the *oscillation* problem mentioned on page 449.) If you were to use the Slowly Changing Dimensions semantic type in this case, there would be a race between the two source systems for each extraction, and (in the worst case), your dimension values could alternate between the two values with every extraction.

Note the following considerations:

- First Dimension Value has the same duplicate **sskey** filtering as Slowly Changing Dimensions.
- New **sskeys** are inserted in both the dimension table and the mapping table. Existing **sskeys** are ignored.
- In the EpiChannel log file, new **sskey** rows are reported in the **Inserted** column. The number of rows in the staging table is reported in the **Processed** column.
- First Dimension Value has the same indexing as Slowly Changing Dimensions.
- The **REAL** column is always equal to the dimension key.
- First Dimension Value has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.
- First Dimension Value cannot be used in Initial jobs.

INITIAL LOAD DIMENSION

The Initial Load Dimension semantic type ignores the current datamart entries. (It is also the fastest semantic type).

Initial Load Dimension loads dimension without regard to any previously existing rows. This semantic type can be used for the initial load of the empty EpiMart, and also to completely reload a dimension, ignoring existing values. Using any other semantic type would require emptying the existing dimension table before beginning the extraction.

Note the following considerations:

- Initial Load Dimension has the same duplicate **sskey** filtering as Slowly Changing Dimensions.
- The existing dimension and mapping tables are ignored; all **sskeys** are imported directly into the dimension and the mapping tables.
- In the EpiChannel log file, new **sskey** rows are reported in the **Inserted** column. The number of rows in the staging table is reported in the **Processed** column.
- Initial Load Dimension has the same indexing as Slowly Changing Dimensions.
- The **REAL** column is always equal to the dimension key.
- Initial Load Dimension has the same UNKNOWN mapping behavior as Slowly Changing Dimensions.
- Initial Load Dimension cannot be used in Incremental or Merge jobs.
- Running Initial Load Dimension on a dimension that has the **indiv** or **group** dimension role invalidates all saved lists that were generated from that dimension.

005270.8155260

REWRITE DIMENSION

The Rewrite Dimension semantic type can be used to make a slowly changing dimension table act like a latest-dimension-value table. Rewrite Dimension first merges values in the staging table into datamart tables using the same logic as the slowly changing dimension semantic. For every **sskey**, it then finds the most recent row with that **sskey** and rewrites every other row with that **sskey** using the dimension column values of the most recent row. No dimension table rows are deleted, so fact tables are still valid. However, the meaning of fact rows may have changed. Since the meaning of fact rows may have changed, a full aggregate build is required on the history tables after running this semantic.

Note that the Rewrite Dimension semantic type can only be used in Rebuild and Non-partitioned jobs.

BACKFEED DIMENSION

The Backfeed Dimension semantic forms a union of the data in the active backfeed export table (P or Q) for a dimension and the data in the main active backfeed table (A or B) for that dimension. The resulting table is then placed in the inactive backfeed table, and the inactive backfeed export table is truncated.

FACT SEMANTIC TYPES

The fact semantic types are Transactional, Transactional/State-like, Transactional/State-like/Force Close, Pipelined, Initial Load Fact, First/Last Fact, Reload Date Fact, and Count Unjoined.

Note: Fact rows with all zero facts are discarded by all semantics except Initial Load Fact.

TRANSACTIONAL

The Transactional semantic type is the simplest means of moving fact-staging data into fact base tables. This semantic type uses the following logic:

- Only rows with **process_key** set to 1 (Transactional) are used; others are discarded.
- For **sskeys** that are already entered in the current-fact base table, all rows in the staging table with that **sskey** and with the same or earlier **date_key** are discarded. If the **sskey** already exists in the fact table, only later dates can be added to the fact table.

Two or more rows with the same **sskey** can be imported into the fact base table if they arrive in the same extraction and the **sskey** either does not already exist, or exists but is dated earlier. (This property is used by the *pseudo-order* approach to the Booking/Shipping problem; see “Transactional/State-like/Force Close,” on page 456.)

- In the EpiChannel log file, all rows added to the fact base table are reported in the **Inserted** column, and the total number of rows in the staging table is reported in the **Processed** column.

Note the following considerations:

- Transactional semantics should be used for event facts; once the fact event has occurred, it can never be modified.
- To reload transactions after they have been loaded into the fact base table, the rows must be deleted from the fact base table, or the fact base table must be truncated. See Reload Date Fact and Initial Load Fact for more information.
- The date comparison is always done with respect to **date_key**, which references the built-in date dimension role. Any user-defined dimension roles that refer to the date dimension are ignored.
- The Transactional semantic type cannot be used in an Initial job.

TRANSACTIONAL/STATE-LIKE

The Transactional/State-like semantic type allows changes to already existing rows in EpiMart. It uses the same logic as the Transactional semantic type, with the addition of the logic described below.

First these three steps occur:

1. Records in the staging table with **process_key** of 2 (state-like) are treated as Orders that can be *differenced* between extractions (a discussion of *differencing* follows).
2. For records in which **process_key** = 2, duplicate **sskeys** with the same **date_key** in the same extraction cause only the highest **ikey** value to be used. Other rows are discarded. This is the same type of filtering that occurs in dimension semantics such as Slowly Changing Dimensions.
3. For an **sskey** with the highest **ikey** (for every set of rows with the same **sskey**, the row with the highest **ikey** takes precedence), if the **date_key** for that staging row is less than the last **date_key** for that **sskey** in the fact base table, the staging row is discarded. In other words, an Order can only be modified on its last reported date, or some time further into the future.

After Steps 1-3, the staging fact columns and dimension values are compared with the current values in the fact table (if any). Adjusting records are created in the fact table so that the fact table now reflects the reported state from the staging table. (This is why the term *state-like* is used.) *Differenced* transactions are invented if the numeric fact columns have changed.

If the dimensionality has changed, then the Order is “debooked” and then “rebooked” with the correct dimensionality.

If the same **sskey** appears in the staging table with more than one **date_key**, then further adjusting transactions are made in the fact base table as appropriate to bring that table in line with the reported staging rows.

Note: By convention, Bookings are entered with positive facts (negative for Returned Orders), whereas Shipments are entered with negative facts (positive for Returned shipments).

When using this semantic type it is difficult to ensure that Backlog calculations will remain consistent when Orders are not completely closed in the source system. Use the Transactional/State-like/Force Close semantic type instead.

Note that the Transactional/State-Like semantic type cannot be used in an Initial job.

TRANSACTIONAL/STATE-LIKE/FORCE CLOSE

This semantic type is equivalent to Transactional/State-like with the following additional logic:

Once a Booked Order is entered into EpiMart, it remains in that state (with an Open Backlog) forever. In normal scenarios, an invoice eventually arrives, which will close the Backlog. However, in some source systems, Booked Orders can be removed from the system completely. If such an Order had been entered previously into EpiMart, then it will remain in an Open Backlog condition forever.

The solution to this problem is to use Transactional/State-like/Force Close, which establishes a “Challenge Protocol” for Open Orders. In this scenario, all Open Bookings must be extracted into the staging table during every extraction. The reason is that the Force Close logic will close out all Open Orders (**sskeys** with non-zero facts in the system) that do not appear in the fact-staging table. Only Booking Transaction types (those with **transtype_key** values between 1 and 99) will be affected in this manner.

To determine whether an order with a given **sskey** is open, the fact values of all rows with that **sskey** are added. This sum is taken over all values of **transtype_key**. If the sum is not zero, then the order is open. If bookings are recorded with positive fact values and shipments are recorded with negative fact values, then the open orders will be exactly those orders for which bookings are not equal to shipments.

When an order is closed by this semantic type, an order entry is constructed with the lowest **transtype_key** value that has been used for this **sskey**. Typically, this **transtype_key** is used for bookings. The fact values in this constructed order entry are those values that result in a sum of zero for the fact table rows with the **sskey** of the order being closed.

When using this semantic type, the methodology that has been found to work in practice involves the use of pseudo-orders. In this methodology, two **sskeys** are used for each order: one **sskey** for the original booking and a second **sskey** for both the shipments and the pseudo-orders. After an order is booked, the following extraction steps are used in every extraction, as long as the order is open:

1. One extraction SQL statement extracts all Open Orders. Fact amounts are the Open amounts (what has not been shipped yet), not the ordered amounts. The transaction type for this statement should be in the Booking range (1...99), the **process_key** value should be 2 (state-like), and the **sskey** value should be the same as that of the original order.
2. The second extraction statement extracts all new shipments. These shipments are recorded with a **process_key** of 1 (Transactional) and with the second **sskey** value. The fact values of these records should be negative for positive shipments. The transaction type should be in the shipment range (101 or greater).
3. The third statement is a restatement of the shipment, but as a Booking (**transtype_key** between 1...99). The **process_key** is still 1 (allowing the Transactional Semantics to import it), but the transaction type is a Booking. The same **sskey** and dimensionality as in the second statement are used. These are the pseudo-orders, since they are actual shipments that are entered as Orders.

The net effect of this methodology is as follows: When a shipment occurs, the open booking quantity (which is extracted in statement 1) is reduced by the number of units shipped. The state-like semantics then creates a difference transaction that reduces the amount of the open order to the new level. At the same time, a pseudo-order (with a new **sskey**) is created for the shipped amount by statement 2, and a (negative) shipment in the same amount is created by statement 3. Note that the convention of recording shipments as negative values ensures that the sum of fact values for the second **sskey** is always zero.

Eventually, when the order is removed from the system, no state-like fact for that **sskey** is generated in statement 1. At this point, if the sum of values for the **sskey** is not zero, then a force-close transaction with values that result in a sum of zero is added to the fact table. Now the fact table has a value of zero for the original **sskey**, and booked and shipped values for the new **sskey** that correspond to the amount actually shipped.

For example, Table 28 illustrates the record for an order that might appear in a staging table.

TABLE 28: ORDER RECORD IN STAGING TABLE

sskey	Cust	Prod	Date	trans	proc	Qty
123a	23	44	4/11/99	1	1	22

Table 29 illustrates how this data is merged into the fact table with transactional logic.

TABLE 29: ORDER RECORD IN FACT TABLE

sskey	Cust	Prod	Date	trans	Qty
123a	23	44	4/11/99	1	22

Table 30 shows the result of extracting the records for 10 items that have been shipped to the customer:

TABLE 30: EXTRACTED FACTS FOR SHIPMENT

sskey	Cust	Prod	Date	trans	proc	Qty
123a	23	44	4/14/99	1	2	12
234b	23	44	4/14/99	101	1	-10
234b	23	44	4/14/99	1	1	10

The first row is a state-like fact for the part of the order that has not yet been shipped. The next two rows are the shipping fact and pseudo-order booking for the part of the order that was shipped. These facts are then merged into the fact table with transactional and state-like logic, as Table 31, on page 459 illustrates.

TABLE 31: SHIPMENT MERGED INTO FACT TABLE

sskey	Cust	Prod	Date	trans	Qty
123a	23	44	4/11/99	1	22
123a	23	44	4/14/99	1	-10
234b	23	44	4/14/99	101	-10
234b	23	44	4/14/99	1	10

If, in the next extraction, there is no record for **sskey** number 123a, then the force-close logic constructs a record to close the order (Table 32).

TABLE 32: FACT TABLE WITH FORCED-CLOSED ORDER

sskey	Cust	Prod	Date	trans	Qty
123a	23	44	4/11/99	1	22
123a	23	44	4/14/99	1	-10
234b	23	44	4/14/99	101	-10
234b	23	44	4/14/99	1	10
123a	23	44	4/17/99	1	-12

The sum of the **Qty** field for all facts with **sskey** 123a is now zero, and the booking and shipping records are in the fact table with **sskey** 234b.

Note that the Transactional/State-Like/Force Close semantic type cannot be used in an Initial job.

PIPELINED

Use the Pipelined semantic type for facts that can exist in several different life-cycle phases, called pipeline states; for example, sales opportunities or support calls facts.

The **transtype_key** field represents the pipeline. Before using this semantic type, you must define the pipeline stages and numbers.

Pipelined generates these transactions:

1. When an **sskey** first enters some pipeline stage (**transtype_key**), a positive fact row is created with that **transtype_key**.
2. If that **sskey** subsequently moves backwards in the pipeline (that is, if the same **sskey** appears with a smaller **transtype** value), then in addition to the new row created by the previous step for the new pipeline stage, a negative “de-booking” transaction is created with the old **transtype_key** minus 1. (This is a loss transaction.)
3. If that **sskey** subsequently moves forwards in the pipeline (that is, if the same **sskey** appears with a larger **transtype** value), then in addition to the new row created in Step 1 for the new pipeline stage, a negative “de-booking” transaction is created with the old **transtype_key** plus 1. (This is a shipment transaction.)

Note: The transtype values for all pipeline stages must be multiples of 3. The +1 and -1 transtype values used in steps 2 and 3 above are created automatically from these values.

For example, assume that the pipeline consists of the following steps: Prospect, Lead, Quote, and Order. You could set up **transtype_key** fields for this pipeline as follows: 303= Prospect, 306 = Lead, 309 = Quote, and 312 = Order. The semantic type creates its own transactions for 302, 304, 305, 307, 308, 310, 311, and 313, which correspond to the forward and backward movements from the various pipeline stages.

To define measures that extract information such as Number of New Leads, you must define transaction types with these new names and keys (using EpiCenter Manager’s Configuration dialog box; see Appendix B, “EpiCenter Configuration.”.)

*Note: The pipelined semantic types ignores the **process_key** field.*

Note that the Pipelined semantic type cannot be used in an Initial job.

INITIAL LOAD FACT

Similar to the Initial Load Dimension, the Initial Load Fact semantic type is the fastest way to load a fact table. It also has the special property that the active current and history tables are ignored so that this semantic type can reload an already-populated fact table. All existing rows, however, are lost. For this reason, Initial Load Fact is usually used during development when an installation is verifying whether a first extraction yields correct data. All data is loaded into the history table (X or Y), and the current table (A or B) is left empty.

Important: For first time extractions, you can use this semantic template in place of all other fact semantic types that load data when each **sskey** is being loaded into the staging table *only once*. This is because upon first extract, when an **sskey** is loaded only once, the special logic of the other semantic types (such as delta inference during Transactional/State-like) does not apply. If the first extraction does require multiple records per **sskey**, such as loading inventory values using Transactional/Statelike, then Initial Fact Load cannot be used.

Note that the Initial Load Fact semantic type cannot be used in Incremental jobs.

INITIAL LOAD FACT A/B

The Initial Load Fact A/B semantic type is the same as the Initial Load Fact semantic type, except that all data is loaded into the A or B table. This semantic type can only be used with the Non-Partitioned job type, and it is intended to be used in datamarts that do not make use of the history (X and Y) tables.

RELOAD DATE FACT

The Reload Date Fact semantic type is a hybrid of the Transactional and Initial Load Fact semantic types. Use it to reload a subset of an existing table in which the reload point is separable by date only.

Reload Date Fact first determines the minimum date that occurs in the fact-staging table. It then copies from the current fact table to the new fact table all existing EpiMart data that occurs with a date key prior to that minimum date.

The fact-staging table is then used to hold all subsequent dates. Thus, a complete load of data must be placed in the fact-staging data from the minimum date forward. When loading the fact-staging table (with SQL statements), use a WHERE clause based on a particular date.

Note that the Reload Date Fact semantic type cannot be used in incremental jobs.

RELOAD DATE FACT A/B

The Reload Date Fact A/B semantic type is the same as the Reload Date Fact semantic type, except that all data is loaded into the A or B table. This semantic type can only be used with the Non-Partitioned job type, and it is intended to be used in datamarts that do not make use of the history (X and Y) tables.

FIRST/LAST FACT

The First/Last Fact semantic type keeps track of the first and last fact values for an **sskey**.

For example, a store may wish to keep track of only the first and last purchases that a customer has made. This can be done with the First/Last Fact semantic type by using the customer ID as the **sskey**.

The First/Last Fact semantic type does not use the transaction type values in the staging table. Instead, it uses the **FIRST** and **LAST** transaction types. By default, the values for **FIRST** and **LAST** are 120 and 121, respectively. performs the following actions:

1. As with all other fact semantics, if two or more staging table rows have the same **sskey** and date values, only the last of these rows is used.
2. For every **sskey** in the staging table, the staging table row with that **sskey** and the earliest date value is found.
 - If the fact table does not contain a row with this **sskey**, then the staging table row is added to the fact table with a **transtype_key** value of **FIRST**.
 - If the fact table contains a row with this **sskey**, a **transtype_key** value of **FIRST**, and a later date, then the values in the fact table row are replaced with the values in the staging table row. The value of **transtype_key** is set to **FIRST**.
 - If the fact table contains a row with this **sskey** and an equal or earlier date, then the staging table row is ignored.
3. For every **sskey** in the staging table, the staging table row with that **sskey** and the latest date value is found.
 - If the fact table does not contain a row with this **sskey**, then the staging table row is added to the fact table with a **transtype_key** value of **LAST**.
 - If the fact table contains a row with this **sskey**, a **transtype_key** value of **LAST**, and an earlier date, then the values in the fact table row are replaced with the values in the staging table row. The value of **transtype_key** is set to **LAST**.
 - If the fact table contains a row with this **sskey** and an equal or greater date, then the staging table row is ignored.

Note the following considerations:

- The First/Last Fact semantic type ignores the **transtype_key** and **process_key** values in the staging table.
- The First/Last Fact semantic type always adds two fact table rows for every new **sskey** in the staging table. If the new **sskey** only appears in one staging table row, than these two fact table rows are identical except for the value of **transtype_key**.
- The First/Last Fact semantic type cannot be used in Incremental or Non-Partitioned jobs.

COUNT UNJOINED

Count Unjoined informs you of the number of rows that are transformed by an outer join when facts are loaded from a staging table to the main fact tables. In order to prevent the loss of rows, outer joins are used to map fact-staging tables to dimension tables. Any unmatched dimension keys in fact-staging tables are automatically set to refer to the special UNKNOWN dimension value. The Count Unjoined semantic counts the number of rows that are transformed in this way and displays the result in the console window.

Note that the Count Unjoined semantic does not change any of the datamart tables.

BACKFEED FACT

The Backfeed Fact semantic forms a union of the data in the active backfeed export table (P or Q) for a fact and the data in the main active backfeed table (A or B) for that fact. The resulting table is then placed in the inactive backfeed table, and the inactive backfeed export table is truncated.

COMPARISON OF SEMANTIC TYPES

This section illustrates the differences between semantic types by means of examples from a fictitious datamart containing customer, product, and order data.

DIMENSION SEMANTICS

Table 33 and Table 34 show a set of rows that might appear in a pair of dimension staging tables after an initial extraction.

TABLE 33: INITIAL CUSTOMER STAGING TABLE

ikey	sskey	iss	Customer_Name	Age	Date
1	bobb	2	Bob Bobster	47	4/23/99
2	lobt	2	Lobby Tableman	15	4/7/99
3	betk	2	Betty Kezer	31	4/15/99

TABLE 34: INITIAL PRODUCT STAGING TABLE

ikey	sskey	iss	Product_Name	Date
1	cool	2	Cool Stuff 2	4/30/99
2	wow	2	Wowzer Pro	4/30/99
3	neat	2	Neato Lite	4/30/99

The **iss** key is a unique identifier for the source system type (the source system type is selected in the data store dialog box). The staging table rows are numbered sequentially by the value in the **ikey** column.

Table 35 and Table 36 show how these tables are loaded into the datamart with an Initial Load Dimension semantic.

TABLE 35: INITIAL CUSTOMER DIMENSION TABLE

key	Customer_Name	Age	Date	key_REAL
2	Bob Bobster	47	4/23/99	2
3	Lobby Tableman	15	4/7/99	3
4	Betty Kezer	31	4/15/99	4

TABLE 36: INITIAL PRODUCT DIMENSION TABLE

key	Product_Name	Date	key_REAL
2	Cool Stuff 2	4/30/99	2
3	Wowzer Pro	4/30/99	3
4	Neato Lite	4/30/99	4

The **key_REAL** column records the first row containing an entry for that **sskey**. In an initial load, the **key_REAL** value is always equal to the **key** value.

Each dimension table also has an **UNKNOWN** row, which always has a **key** value of 1. The **UNKNOWN** row is not illustrated in these tables.

The datamart also has mapping tables that map **sskeys** to dimension table **keys**. For example, the Customer mapping table might look like the Table 37.

TABLE 37: INITIAL CUSTOMER MAPPING TABLE

iss	sskey	key
2	bobb	2
2	lobt	3
2	betk	4

In a subsequent extraction, you might load the following new data into the Customer staging table (Table 38).

TABLE 38: SECOND CUSTOMER STAGING TABLE

ikey	sskey	iss	Customer_Name	Age	Date
1	jrd	2	J.R. Dobbs	23	5/10/99
2	lobt	2	Lobby Tableman	16	5/5/99
3	jrd	2	J.R. Dobbs	20	5/2/99
4	bobb	2	Bob Bobster	47	4/23/99

This data can be merged into the dimension tables in different ways, depending on your choice of dimension semantic.

Note that all dimension semantics ignore the first row, since there is another staging table row with the same **sskey** and a higher **ikey** value. All semantics also ignore the last row, since it is a duplicate of a row that is already in the dimension table (row 2).

09625518.072500

SLOWLY CHANGING DIMENSIONS

If the new data is merged with a Slowly Changing Dimensions semantic, then the revised dimension table looks like Table 39.

TABLE 39: CUSTOMER DIMENSION TABLE AFTER SLOWLY CHANGING DIMENSIONS SEMANTIC

key	Customer_Name	Age	Date	key_REAL
2	Bob Bobster	47	4/23/99	2
3	Lobby Tableman	15	4/7/99	3
4	Betty Kezer	31	4/15/99	4
5	Lobby Tableman	16	5/5/99	3
6	J.R. Dobbs	20	5/2/99	6

The new data for Lobby Tableman is appended to the dimension table. The **key_REAL** value of the new row points to the first row with that **sskey**, which is row 3.

The revised mapping table looks like Table 40.

TABLE 40: CUSTOMER MAPPING TABLE AFTER SLOWLY CHANGING DIMENSIONS SEMANTIC

iss	sskey	key
2	bobb	2
2	lobt	5
2	betk	4
2	jrd	6

The mapping table now associates the **sskey** for Lobby Tableman with the new row, and all new facts that refer to customer **sskey** lobt will have a Customer foreign key of 5. However, previously extracted facts that referred to **sskey** lobt continue to have a Customer foreign key of 3.

09625518-072500

FIRST DIMENSION VALUE

If the new data is merged with a First Dimension Value semantic, then the revised dimension table looks like Table 41.

TABLE 41: CUSTOMER DIMENSION TABLE AFTER FIRST DIMENSION VALUE SEMANTIC

key	Customer_Name	Age	Date	key_REAL
2	Bob Bobster	47	4/23/99	2
3	Lobby Tableman	15	4/7/99	3
4	Betty Kezer	31	4/15/99	4
5	J.R. Dobbs	20	5/2/99	5

The data for the new customer has been appended to the table, and the revised data for Lobby Tableman has been discarded.

The revised mapping table looks like Table 42

TABLE 42: CUSTOMER MAPPING TABLE AFTER FIRST DIMENSION VALUE SEMANTIC

iss	sskey	key
2	bobb	2
2	lobt	3
2	betk	4
2	jrd	5

LATEST DIMENSION VALUE

If the new data is merged with a Latest Dimension Value semantic, then the revised dimension table looks like Table 43.

TABLE 43: CUSTOMER DIMENSION TABLE AFTER LATEST DIMENSION VALUE SEMANTIC

key	Customer_Name	Age	Date	key_REAL
2	Bob Bobster	47	4/23/99	2
3	Lobby Tableman	16	5/5/99	3
4	Betty Kezer	31	4/15/99	4
5	J.R. Dobbs	20	5/2/99	5

The data for the new customer has been appended to the table, and the revised data for Lobby Tableman has been used to update row 3. The revised mapping table looks like Table 44.

TABLE 44: CUSTOMER MAPPING TABLE AFTER LATEST DIMENSION VALUE SEMANTIC

iss	sskey	key
2	bobb	2
2	lobt	3
2	betk	4
2	jrd	5

REWRITE DIMENSION

If the new data is merged with a Rewrite Dimension semantic, then the revised dimension table looks like Table 45.

TABLE 45: CUSTOMER DIMENSION TABLE AFTER REWRITE DIMENSION SEMANTIC

key	Customer_Name	Age	Date	key_REAL
2	Bob Bobster	47	4/23/99	2
3	Lobby Tableman	16	5/5/99	3
4	Betty Kezer	31	4/15/99	4
5	Lobby Tableman	16	5/5/99	3
6	J.R. Dobbs	20	5/2/99	6

As with the Slowly Changing Dimensions semantic, the revised data for Lobby Tableman is appended to the table. In addition, the old entry (in row 3) is updated with the new information.

The revised mapping table looks like Table 46.

TABLE 46: CUSTOMER MAPPING TABLE AFTER REWRITE DIMENSION SEMANTIC

iss	sskey	key
2	bobb	2
2	lobt	5
2	betk	4
2	jrd	6

FACT SEMANTICS

After the initial extraction, you might have the following data in the fact-staging table (Table 47).

TABLE 47: INITIAL ORDER STAGING TABLE

ikey	sskey	iss	Cust	Prod	Date	trans	proc	Qty
1	41N	2	lobt	neat	4/22/99	1	1	4
2	65R	2	bobb	neat	4/8/99	1	1	6
3	57Q	2	bobb	wow	4/23/99	1	1	3
4	48Z	2	lobt	cool	4/7/99	1	1	7
5	91B	2	lobt	wow	4/1/99	1	1	1
6	46C	2	betk	cool	4/19/99	1	1	2
7	55T	2	bobb	cool	4/17/99	1	1	3

Here **Cust** is the Customer **sskey**, **Prod** is the Product **sskey**, **trans** is the transtype key, **proc** is the process key, and **Qty** is the quantity ordered.

When this data is loaded into the datamart with an initial load fact semantic, you get a datamart table like Table 48.

TABLE 48: INITIAL ORDER FACT TABLE

seq	sskey	iss	Cust	Prod	Date	trans	Qty
1	41N	2	3	4	4/22/99	1	4
2	65R	2	2	4	4/8/99	1	6
3	57Q	2	2	3	4/23/99	1	3
4	48Z	2	3	2	4/7/99	1	7
5	91B	2	3	3	4/1/99	1	1
6	46C	2	4	2	4/19/99	1	2
7	55T	2	2	2	4/17/99	1	3

Here **Cust** and **Prod** are the integer foreign keys for the dimension table rows that correspond to a fact, and **seq** is a sequential numbering of the fact table rows.

In a subsequent extraction, you might load the following new data into the Order staging table (Table 49).

TABLE 49: NEW ORDER STAGING TABLE

ikey	sskey	iss	Cust	Prod	Date	trans	proc	Qty
1	57Q	2	bobb	wow	4/23/99	1	2	3
2	48Z	2	lobt	cool	4/7/99	1	2	5
3	46C	2	betk	cool	4/19/99	1	1	1
4	11L	2	betk	wow	4/7/99	1	1	1
5	65R	2	bobb	neat	4/8/99	1	2	7
6	71Z	2	jrd	neat	5/8/99	1	2	5
7	41N	2	lobt	neat	4/24/99	1	1	3
8	26F	2	jrd	wow	5/14/99	1	1	10
9	57Q	5	jrd	cool	4/20/99	1	1	3
10	55T	2	bobb	neat	4/17/99	1	2	3

This data can be merged into the fact tables in different ways, depending on your choice of both fact and dimension semantics.

TRANSACTIONAL

If the new data is merged with a Transactional semantic, then all transactional facts (that is, facts with a **process_key** value of 1) are added to the fact table, unless the fact table already contains a fact with the same **sskey** and the same date or a later date. State-like facts (that is, facts with a **process_key** value of 2) are ignored.

The exact structure of the revised fact table depends on the semantic that was used to update the Customer dimension. If the Customer dimension is updated with a First Dimension Value or Latest Dimension Value semantic, then the fact table looks like Table 50.

TABLE 50: ORDER FACT TABLE, TRANSACTIONAL REVISION 1

seq	sskey	iss	Cust	Prod	Date	trans	Qty
1	41N	2	3	4	4/22/99	1	4
2	65R	2	2	4	4/8/99	1	6
3	57Q	2	2	3	4/23/99	1	3
4	48Z	2	3	2	4/7/99	1	7
5	91B	2	3	3	4/1/99	1	1
6	46C	2	4	2	4/19/99	1	2
7	55T	2	2	2	4/17/99	1	3
8	11L	2	4	3	4/1/99	1	1
9	41N	2	3	4	4/24/99	1	3
10	26F	2	5	3	5/14/99	1	10
11	57Q	5	5	2	4/20/99	1	3

Note the following considerations:

- No staging table rows with a **process_key** value of 2 are added to the fact table.
- If a staging table row has the same **sskey** as a fact table row and the same date value as or an earlier date value than some row of the fact table, then that staging table row is not added to the fact table. For example, row 3 of the staging table was not added to the fact table because it has the same **sskey** and the same date as row 6 of the fact table.

- If a staging table row has the same **sskey** value but a different **iss** value, then it is treated as if it had a different **sskey** value. For example, row 9 of the staging table has the same **sskey** value as row 3 of the fact table, but a different **iss** value. Therefore, it is added to the fact table, even though it has an earlier date.
- All other staging table rows are added to the fact table.

If the Customer dimension is updated with a Slowly Changing Dimensions or Rewrite Dimension semantic, then the fact table looks like Table 51.

TABLE 51: ORDER FACT TABLE, TRANSACTIONAL REVISION 2

seq	sskey	iss	Cust	Prod	Date	trans	Qty
1	41N	2	3	4	4/22/99	1	4
2	65R	2	2	4	4/8/99	1	6
...
8	11L	2	4	3	4/1/99	1	1
9	41N	2	5	4	4/24/99	1	3
10	26F	2	6	3	5/14/99	1	10
11	57Q	5	6	2	4/20/99	1	3

The important difference here is that the new order for Lobby Tableman (row 7 of the staging table, row 9 of the fact table) is entered with a customer key value that points to the dimension table row with updated values (row 5).

The existing orders for Lobby Tableman still have customer keys that point to the dimension table row with old values (row 3).

TRANSACTIONAL/STATE-LIKE

If the new data is merged with a Transactional/State-Like semantic, then transactional facts are added as they are with a Transactional semantic type. State-like facts (that is, facts with a **process_key** value of 2) are assumed to reflect the current state of the fact quantities for a given **sskey**. If the fact table values do not match those in the state-like fact, then transactions are added to the fact table to adjust those values.

The structure of the revised fact table again depends on the semantic that was used to update the Customer dimension.

If the Customer dimension is updated with a First Dimension Value or Latest Dimension Value semantic, then the fact table looks like Table 52

TABLE 52: ORDER FACT TABLE, TRANSACTIONAL/STATE-LIKE REVISION 1

(1 OF 2)

seq	sskey	iss	Cust	Prod	Date	trans	Qty
1	41N	2	3	4	4/22/99	1	4
2	65R	2	2	4	4/8/99	1	6
3	57Q	2	2	3	4/23/99	1	3
4	48Z	2	3	2	4/7/99	1	7
5	91B	2	3	3	4/1/99	1	1
6	46C	2	4	2	4/19/99	1	2
7	55T	2	2	2	4/17/99	1	3
8	48Z	2	3	2	4/7/99	1	-2
9	11L	2	4	3	4/1/99	1	1
10	65R	2	2	4	4/8/99	1	1
11	71Z	2	5	4	5/8/99	1	5
12	41N	2	3	4	4/24/99	1	3
13	26F	2	5	3	5/14/99	1	10

TABLE 52: ORDER FACT TABLE, TRANSACTIONAL/STATE-LIKE REVISION 1

(2 OF 2)

seq	sskey	iss	Cust	Prod	Date	trans	Qty
14	57Q	5	5	2	4/20/99	1	3
15	55T	2	2	2	4/17/99	1	-3
16	55T	2	2	4	4/17/99	1	3

Note the following considerations:

- Staging table rows with a **process_key** value of 1 are added to the fact table as with the Transactional semantic type.
- If a staging table row with a **process_key** value of 2 has the same dimensionality (that is, the same dimension key values) and the same **sskey** value as an existing fact table row, but a different fact value, then a difference row is added to the fact table. For example, row 5 of the staging table has the same **sskey** and dimensionality values as row 2 of the fact table, but the order quantity is 7 rather than 6. Row 10 of the revised fact table therefore has an order for a single unit, so that the orders for **sskey** 65R add up to 7. Similarly, row 8 updates the value in row 4.
- If a staging table row with a **process_key** value of 2 has the same **sskey** value as an existing fact table row but different dimensionality, then the old order is debooked and the new order is booked. For example, row 10 of the staging table has the same **sskey** as row 7 of the fact table, but a different product dimension key. Therefore, row 15 of the revised fact table debooks the order with the old dimension values, and row 16 rebooks it with the new dimension values.
- If a staging table row with a **process_key** value of 2 has the same dimensionality, the same **sskey** value, and the same fact values as an existing fact table row, then that staging table row is ignored. For example, row 1 of the staging table duplicates row 3 of the fact table, so it is ignored.

If the Customer dimension is updated with a Slowly Changing Dimensions or Rewrite Dimension semantic, then the fact table looks like Table 53.

TABLE 53: ORDER FACT TABLE, TRANSACTIONAL/STATE-LIKE REVISION 2

seq	sskey	iss	Cust	Prod	Date	trans	Qty
1	41N	2	3	4	4/22/99	1	4
2	65R	2	2	4	4/8/99	1	6
3	57Q	2	2	3	4/23/99	1	3
4	48Z	2	3	2	4/7/99	1	7
5	91B	2	3	3	4/1/99	1	1
6	46C	2	4	2	4/19/99	1	2
7	55T	2	2	2	4/17/99	1	3
8	48Z	2	3	2	4/7/99	1	-7
9	48Z	2	5	2	4/7/99	1	5
10	11L	2	4	3	4/1/99	1	1
...
16	55T	2	2	2	4/17/99	1	-3
17	55T	2	2	4	4/17/99	1	3

The difference here is in rows 8 and 9. Row 2 of the staging table refers to the same customer as row 4 of the fact table. However, since the data for Lobby Tableman has changed and the dimension has been updated with a Slowly Changing Dimensions or Rewrite Dimension semantic, a new row with revised data has been added to the dimension table. Since the state-like fact refers to a different dimension table row than that referred to by the initial transactional fact, it is treated as a state-like fact with changed dimensionality. Therefore, the original order is debooked and the new order is rebooked.

TRANSACTIONAL/STATE-LIKE/FORCE CLOSE

If the new data is merged with a Transactional/State-Like/Force Close semantic, then new data is merged in the same way as with Transactional/State-Like. In addition, the Force Close logic is applied to facts with **sskeys** that do not appear in the staging table. When a fact table **sskey** does not appear in the staging table, and there is at least one row in the fact table with that **sskey** and an Order transaction type (that is, with a **transtype_key** value in the range 1-99), then that **sskey** is forced closed. For such an **sskey**, all facts with the **sskey** (regardless of transtype) are added. If this sum is not zero, then a new fact that forces the sum to be zero is added to the fact table. Note that if all fact table rows with a given **sskey** value have **transtype_key** values greater than 99, then that fact is not affected by the force close logic.

The Transactional/State-Like/Force Close semantic type is most commonly used with the pseudo-order approach described on page 457, but it can be applied to any fact and staging tables. When applied to the example fact and staging tables in this section, the exact structure of the revised fact table again depends on the semantic that was used to update the Customer dimension.

If the Customer dimension is updated with a First Dimension Value or Latest Dimension Value semantic, then the fact table looks like Table 54.

TABLE 54: ORDER FACT TABLE, TRANSACTIONAL/STATE-LIKE/FORCE CLOSE REVISION 1

seq	sskey	iss	Cust	Prod	Date	trans	Qty
1	41N	2	3	4	4/22/99	1	4
...
5	91B	2	3	3	4/1/99	1	1
6	46C	2	4	2	4/19/99	1	2
7	55T	2	2	2	4/17/99	1	3
...
15	55T	2	2	2	4/17/99	1	-3
16	55T	2	2	4	4/17/99	1	3
17	91B	2	3	4	4/1/99	1	-1

Note the following considerations:

- Since there is no row in the staging table with **sskey** 91B, row 17 of the revised fact table de-books the order in row 5.
- The order in row 6 is not de-booked, because row 3 of the staging table has the same **sskey**.
- All staging table rows are treated as they would be with a Transactional/State-Like semantic type. In particular, row 3 of the staging table is not added to the fact table, because its date is before that of row 6 of the fact table.

If the Customer dimension is updated with a Slowly Changing Dimensions or Rewrite Dimension semantic, then the fact table looks like Table 55.

TABLE 55: ORDER FACT TABLE, TRANSACTIONAL/STATE-LIKE/FORCE CLOSE REVISION 2

seq	sskey	iss	Cust	Prod	Date	trans	Qty
1	41N	2	3	4	4/22/99	1	4
...
7	55T	2	2	2	4/17/99	1	3
8	48Z	2	3	2	4/7/99	1	-7
9	48Z	2	5	2	4/7/99	1	5
10	11L	2	4	3	4/1/99	1	1
...
16	55T	2	2	2	4/17/99	1	-3
17	55T	2	2	4	4/17/99	1	3
18	91B	2	3	3	4/1/99	1	-1

The only difference here is that rows 8 and 9 de-book and re-book the order with **sskey** 48Z, as with the Transactional/State-Like semantic type.

RELOAD DATE FACT

If the new data is merged with a Reload Date Fact semantic, then all facts in the staging table are placed into a new fact table. In addition, all facts from the old fact table that have a date that is earlier than the earliest date in the new fact table are appended to the new fact table.

The exact structure of the revised fact table again depends on the semantic that was used to update the Customer dimension.

If the Customer dimension is updated with a First Dimension Value or Latest Dimension Value semantic, then the fact table looks like Table 56.

TABLE 56: ORDER FACT TABLE, RELOAD DATE FACT REVISION 1

seq	sskey	iss	Cust	Prod	Date	trans	Qty
1	57Q	2	2	3	4/23/99	1	3
2	48Z	2	3	2	4/7/99	1	5
3	46C	2	4	2	4/19/99	1	1
4	11L	2	4	3	4/7/99	1	1
5	65R	2	2	4	4/8/99	1	7
6	71Z	2	5	4	5/8/99	1	5
7	41N	2	3	4	4/24/99	1	3
8	26F	2	5	3	5/14/99	1	10
9	57Q	5	5	2	4/20/99	1	3
10	55T	2	2	4	4/17/99	1	3
11	91B	2	3	3	4/1/99	1	1

This is essentially the staging table with older fact-table data appended.

If the Customer dimension is updated with a Slowly Changing Dimensions or Rewrite Dimension semantic, then the fact table looks like Table 57.

TABLE 57: ORDER FACT TABLE, RELOAD DATE FACT REVISION 2

seq	sskey	iss	Cust	Prod	Date	trans	Qty
1	57Q	2	2	3	4/23/99	1	3
2	48Z	2	5	2	4/7/99	1	5
3	46C	2	4	2	4/19/99	1	1
4	11L	2	4	3	4/7/99	1	1
5	65R	2	2	4	4/8/99	1	7
6	71Z	2	6	4	5/8/99	1	5
7	41N	2	5	4	4/24/99	1	3
8	26F	2	6	3	5/14/99	1	10
9	57Q	5	6	2	4/20/99	1	3
10	55T	2	2	4	4/17/99	1	3
11	91B	2	3	3	4/1/99	1	1

The only difference here is in the dimension keys for the new values. Notice that the new entries for Lobby Tableman have key 5, while the entry (in row 11) that was imported from the old fact table still has a key value of 3.

JOB TYPES AND SEMANTICS

The choice of job type limits the semantics that can be used. The permitted semantics for each job type are listed below.

INCREMENTAL JOBS

The permitted dimension semantics for an Incremental job are:

- Slowly Changing Dimension
- First Dimension Value
- Backfeed Dimension

The permitted fact semantics for an Incremental job are:

- Transactional
- Transactional/State-Like
- Transactional/State-Like/Force Close
- Pipelined
- Backfeed Fact
- Count Unjoined

INITIAL JOBS

The permitted dimension semantics for an Initial job are:

- Initial Load Dimension
- Backfeed Dimension

The permitted fact semantics for an Initial job are:

- Initial Load Fact
- Reload Date Fact
- First/Last Fact

09525518.072500

- Backfeed Fact
- Count Unjoined

MERGE JOBS

The permitted dimension semantics for a Merge job are:

- Slowly Changing Dimension
- First Dimension Value
- Backfeed Dimension

The permitted fact semantics for a Merge job are:

- Transactional
- Transactional/State-Like
- Transactional/State-Like/Force Close
- Pipelined
- Initial Load Fact
- Reload Date Fact
- First/Last Fact
- Backfeed Fact
- Count Unjoined

REBUILD JOBS

The permitted dimension semantics for a Rebuild job are:

- Slowly Changing Dimension
- First Dimension Value
- Latest Dimension Value
- Rewrite Dimension

- Initial Load Dimension
- Backfeed Dimension

The permitted fact semantics for a Rebuild job are:

- Transactional
- Transactional/State-Like
- Transactional/State-Like/Force Close
- Pipelined
- Initial Load Fact
- Reload Date Fact
- First/Last Fact
- Backfeed Fact
- Count Unjoined

NON-PARTITIONED JOBS

The permitted dimension semantics for a Non-Partitioned job are:

- Slowly Changing Dimension
- First Dimension Value
- Latest Dimension Value
- Rewrite Dimension
- Initial Load Dimension
- Backfeed Dimension

The permitted fact semantics for a Non-Partitioned job are:

- Transactional
- Transactional/State-Like
- Transactional/State-Like/Force Close
- Pipelined

- Initial Load Fact A/B
- Reload Date Fact A/B
- Backfeed Fact
- Count Unjoined

09625518 072500

09625518:072500

EXPORT/IMPORT OF METADATA

All of the control information for an EpiCenter datamart is stored in a metadata repository called EpiMeta. This repository takes the form of a transactional, relational database composed of numerous tables and an extensive set of declarative referential integrity constraints. EpiPhany provides the EpiCenter Manager administrative interface for configuring this metadata.

EpiPhany also provides a metadata Export/Import utility for backing up the definition of an EpiCenter and moving metadata from one EpiCenter to another. To use the Export/Import tool properly, you should understand the basic metadata concepts discussed in this appendix.

METADATA OVERVIEW

All of the user-configurable metadata tables have integer primary keys. These non-natural keys are provided by the database engine when a metadata row is inserted. The value of the primary key itself has no intrinsic meaning. All relationships to this inserted row are made via this integer. For example, the relationship between a dimension role and the base dimension from which it is derived is represented in EpiMeta by an integer column called `dim_base_key`.

The Access Export database does not simply contain a copy of the metadata tables being exported for this reason: If data were copied to the Export file, then when this same information is imported into a new EpiMeta, the integer primary key values in the Export file might clash with already existing primary keys in the target EpiMeta. Therefore, the Access database uses an EpiMeta-independent representation of metadata. See "Export File Format," on page 494 for a description of these Access database tables.

EpiMeta contains many tables, all of which are inter-related. In order to export only a portion of the metadata at a time, the Export machinery must decide where to stop exporting—otherwise, the entire metadata must be exported with each operation.

When using the **Export** command of EpiCenter Manager, you can select which part of the metadata to export. The Exporting Metadata dialog box is organized by related metadata: **Schema**, **Measures**, **Extraction**, **Presentation**, **Security**, **Storage**, and **Runtime**.

Reimporting objects does not delete references to those objects. For example, a measure definition can be reimported without deletion of the measure mappings in Web pages that point to that measure. If the Import machinery finds these references, then the same relationships are established in the new EpiMeta as the ones that were exported. However, if the Import machinery does not find these measures by name, the measure mapping information is lost upon import.

REPLACING EXISTING METADATA ON IMPORT

When importing metadata into an existing EpiMeta, the import machinery detects an attempt to overwrite existing metadata. Existing top-level objects are overwritten in place. Their children are deleted, however, before being recreated. The definition of “existing” is usually based on a unique name column for one of the metadata tables. For instance, Web pages must have unique names, so an attempt to import a Web page with the same name as an existing one results in a warning message (unless the **Always Replace Existing Data** option has been selected previously).

TRANSACTION TYPES

The Transaction Types tab of the EpiCenter Configuration dialog box allows you to rename transaction types. However, unlike most other metadata, the name of the transaction type is used in dimension tables, not the key. This leads to problems when you try to export and then import metadata. If you rename an existing transaction type, that change in metadata is propagated to the local metadata tables. When you export transaction-type definitions and then import them to another instance, EpiCenter Manager does not overwrite the existing transaction types—it adds new ones.

Data that references the old transaction type continues to do so after importing the new type. If you have renamed existing transaction types in the source system for your metadata, you must restore the original names of each transaction type before exporting metadata. EpiPhany recommends that you use the `$$TRANSTYPE` macro whenever you refer to a transaction type in an SQL statement. Refer to Appendix A, “EpiPhany Macros,” for details.

ACTUALS AND AGGREGATE METADATA

The **Export All** command in EpiCenter Manager does not export the entire contents of EpiMeta. The options for Actuals and Aggregates are omitted by default from this export. This is because these sections of metadata are “derived” metadata: the Adaptive Schema Generator produces Actual table metadata, and Aggbuilder produces aggregate metadata. As a result, a new EpiCenter can be constructed using the Export All metadata by rebuilding the schema, re-extracting, and re-running Aggbuilder.

To “clone” an EpiMeta in such a way that EpiMart itself does not need any modification when it is used with this new EpiMeta, you must select the **Runtime** options for **Actuals** and **Aggs** in the Exporting Metadata dialog box.

Warning: *An Export All operation on a running system, followed by a re-import of that metadata causes all Actual and Aggregate metadata to be lost.*

EXPORT FILE FORMAT

Each Microsoft Access Export database has the same schema. This schema can be thought of as a metaschema for representing relational data. For a description of these tables, see Table 58.

To modify the row contents contained in an Export file, edit the **Export_col** table, which is simply a collection of name/value pairs for columns.

TABLE 58: EXPORT TABLES

Table Name	Description
Export_tbl	One row per metadata table being exported.
Export_row	One row per metadata row being exported.
Export_col	One row per column per row of metadata being exported. Only non-relationship columns are contained in this table.
Export_rel	One row per relationship between two rows of metadata. Can be a relationship between two rows contained in the Export file, or between one row in the Export file and one reference to a row in a foreign EpiMeta.
Export_status	Header information about the Export file.
Rel_parent	A reference to a metadata row in the foreign EpiMeta.

TROUBLESHOOTING

This chapter describes the E.piphany error conditions and error messages and suggests action you can take to resolve problems.

Warning: Using Registry Editor incorrectly can cause serious problems that may require you to reinstall your operating system. Microsoft cannot guarantee that problems resulting from the incorrect use of Registry Editor can be solved. Use Registry Editor at your own risk.

For information about how to edit the Registry, view the “Changing Keys And Values” Help topic in Registry Editor (**Regedit.exe**), or the “Add and Delete Information in the Registry” and “Edit Registry Data” Help topics in **Regedt32.exe**.

Note: E.piphany suggests that you back up the Registry before you edit it.

Any number of problems can arise when either your AppServer host or the database-server host machine runs out of disk space. Whenever you experience unfamiliar or intermittent errors, check to ensure that your host machines have an adequate amount of free disk space.

If the corrective actions that are presented in an error message that the AppServer displays, or in the discussion of problems presented in this chapter, do not correct the problem, please send e-mail with a description of the problem and the steps you have taken to correct it to E.piphany Customer Support at support@epiphany.com.

APPLICATION SERVER ERROR MESSAGES

Note: If the Application Server fails to start, you can run the Scrutiny debugging tool in EpiCenter Manager to diagnose the problem. See “Running the Scrutiny Debugging Tool,” on page 299.

The following error messages are described on the indicated pages:

- “User Cannot Log In,” on page 496
- “Internal Windows NT Error,” on page 498
- “EpiQuery Engine Database Connection Open Failure Exception,” on page 498
- “Charts Do Not Display,” on page 498
- “GIF Images Fail to Display on Web Pages,” on page 499
- “Result Page Error: Extraction Date Unknown,” on page 500
- “Web Server Message: Object Not Found,” on page 500
- “Browser Crashes When Retrieving Results from Application Server,” on page 502
- “Refresh Program Fails,” on page 503
- “Application Log Full Error,” on page 503

USER CANNOT LOG IN

There are three types of failure associated with an unsuccessful login to the Application Server:

1. The Application Server generates a critical exception indicating an application error, or an error related to the NT security domain).

If this case, an error message appears in the browser with a link to the log that contains the stack trace for the error. Read the error description. Generally, it relates to the way NT domain security is configured or set up at this point. For example, the error might say that the domain controller could not be reached.

2. An invalid login message displays even though there is a valid username and password.

This error may result when the search for the username/password occurs on the wrong machine. For example, suppose a user *xyz* existed on both the machine local to the Application Server and the primary domain controller.

The order in which the user *xyz* is searched is as follows:

- the local SAM
- the primary domain
- the trusted domains

Thus, user *xyz* in the local SAM will be found first, even though the username/password combination could have been for the user *xyz* that exists in the primary domain or in a trusted domain.

To solve this problem, further specify the username by including the domain name before the username; for example, *EPIPHANY\xyz*. In general, specify the full user's name if the user's browser displays an invalid login message.

3. Authentication was successful, but the user is not allowed to use the E.piphany system.

For a user to have access to E.piphany System, he or she must be a member of at least one E.piphany group after a sync-up process.

Use EpiCenter Manager to check group memberships after you receive this error message. If the old group memberships were removed, then this error has occurred because in the NT domain, the user is not a member of the NT groups that have the corresponding groups marked Synchronize in EpiCenter Manager.

Also, check that the username used to log in matches the username in EpiCenter Manager. If the username in EpiCenter Manager is prefixed with domain name other than the one that the actual NT user is a member of, then the login could fail and return an error message to this effect. Specifying the full username upon log in may fix this problem.

INTERNAL WINDOWS NT ERROR

This error results when the E.piphany Application Server cannot be started as a service. The exact error message that was returned from the E.piphany Application Server is logged in the Windows NT Event Log.

To locate this error message in the Event Log:

1. Go to **Start\Programs\Administrative Tools (Common)\Event Viewer**.
2. Click the **Log** menu and select the **Application**.
3. Double-click the appropriate event.

All E.piphany Application Server events have the source **EpiAppServer**.

To solve this problem, first stop any running Application Server services. Then log onto Windows NT as an administrator and re-install the E.piphany software. If this does not solve the problem, start the Application Server from the command line as described in “Monitoring the Application Server,” on page 360. The console output should describe what is wrong.

EPIQUERY ENGINE DATABASE CONNECTION OPEN FAILURE EXCEPTION

An **EpiQueryEngineDBConnOpenFailureException** from the EpiQueryEngine means that the Application Server is having difficulty connecting to the datamart. Take these steps to correct this problem:

- Make sure that the username and password for the EpiMeta database are set correctly in the Windows Registry.
- Make sure that the database name and server are also configured properly (again, in the Registry).
- Make sure that TCP/IP connectivity is properly configured.:

CHARTS DO NOT DISPLAY

If there is a broken link to an image, or no image appears, or an image with an error message appears instead of the chart, there is a charting problem. Please contact E.piphany Customer Support.

GIF IMAGES FAIL TO DISPLAY ON WEB PAGES

If GIF images do not appear on your Web pages, do the following:

1. Make sure that your Web server has an alias for your *instance_name* that points to a valid directory. If you performed a normal installation, then all Web files should be located in the directory:

C:\Program Files\Epiphany\instance_name\web\WWWROOT

and GIF files should be located in the **images** subdirectory:

If your Web server is IIS 3.0, you can find the aliases by opening up the IIS Internet Service Manager (normally this is located in your Start\Microsoft Internet Information Server menu) and selecting the Directory tab. Make sure that there is an entry for *instance_name* that has a valid directory.

For the IIS 4.0 Web server, open the Windows NT Option Pack\Microsoft Internet Server\Internet Service Manager. Aliases are listed in the IIS Management Console.

2. Make sure the **WWWROOT\images** directory has the GIFs in it. If you are missing a few GIFs, then you need to reinstall your Application Server or copy the GIFs from a different instance.
3. Check the BASE HREF tag that is defined in the source of the page. In your browser, try to view the source for this HTML page. Look for the BASE HREF tag at the top of the page. Note what it is and make sure that it is a valid alias using the procedure above.
4. Make sure your Web server is serving pages and that your browser is not displaying cached HTML. Clear the caches (Memory and Disk) on your browser, close your browser, and try to access the URL again. Also, try referencing another URL from your Web server to make sure that it is running.

Technical Note: All of the GIFs on an Application Server-generated page are referenced from the **images** directory, which is relative to the BASE HREF specified at the top of the page in a META tag. The *instance_name* is derived from the URL that you use to access the Application Server. It is used throughout the system to read the correct Registry entries and to generate the correct URLs.

RESULT PAGE ERROR: EXTRACTION DATE UNKNOWN

The `last_extract_date` is a field that is kept in the EpiMeta database. It is used to keep track of the date displayed on the top of all reports as the date of the last extraction. It is normally populated by the extraction SQL entered in the End of Extraction extraction group in the EpiCenter Manager. It can also be populated by EpiCenter Manager via the Configuration dialog box. This field must be entered in one of two very strict formats. The default format is `mm/dd/yyyy`; for example: 01/14/1998.

The Application Server applies the following logic to parse the date:

1. If the field has more than 10 characters, then parse it using the pattern `mm/dd/yyyy hh:mm:ss`. Otherwise, use the pattern `mm/dd/yyyy`.
2. If the parse fails, then use `{extraction date unknown}`.

In addition, the date that is displayed at the top of the report always has a time zone. The time zone is printed based on the default time zone of the machine on which the Application Server is running. The date that is being displayed is also taken into consideration. For example, the date 12/20/1997 displays as **December 20, 1997 PST** if the Application Server was running on a machine in California. However, the day 05/12/1998 will display as **May 12, 1998 PDT** on the same machine since Daylight Savings time took affect in April.

Note: EpiCenter Manager does not allow the user to enter a date in the format `mm/dd/yyyy hh:mm:ss`. Only the SQL in the extraction job can enter dates of this format, or you can manually arrange this via `isql`.

WEB SERVER MESSAGE: OBJECT NOT FOUND

If you installed the E.piphany software using the E.piphany installer, you can access your Web server through this type of URL:

`http://hostname/scripts/instance_name/Epiphany.dll`

If you receive an object not found error message, follow these steps:

1. Start and stop the Web server. If this does not solve the problem, go to the next step.
2. Verify the Web server is serving pages.

Note: *Make sure that your browser is not just returning cached HTML pages by clearing your memory and disk cache before testing.*

Try to access other URLs from the same *machinename*. Try to access other static HTML files that are installed as a part of the Application Server installation, such as:

`http://machinename/instance_name/WWWRoot/help/WebPages.html.`

3. If this does not work, try accessing any other file that the Web server should be serving. Consult the Internet Service Manager for the names of other aliases that the Web server should be serving, and then try to access these aliases with your browser.

In most cases, your Web server searches the

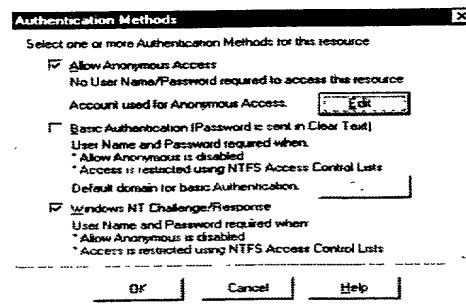
C:\Inetpub\scripts\instance_name directory to find the **Epiphany.dll** program. Make sure that there is such a directory on your machine, and that the **Epiphany.dll** file is in that directory.

4. Check the file permissions for the **Epiphany.dll** file.

First, make sure that the account IIS uses for anonymous logins has file access permissions for the **Epiphany.dll** file. Go to the IIS 3.0 Internet Service Manager and look at the Anonymous Login account box. In IIS 4.0, right-click the name of the machine and choose Properties. Select the Directory Security tab.

In the Authentication Methods dialog box, select **Allow Anonymous Access** and click the **Edit** button to modify the account used for this purpose. Make sure the user and password are correct.

FIGURE 116: AUTHENTICATION METHODS DIALOG BOX



To check file permissions, open the Windows NT Explorer window and right-click the **Epiphany.dll** file in the **./scripts/instance_name** directory specified above. Go to Properties and open the Security tab. Click **View Permissions** and make sure that the account that you used for Anonymous Web Login has access to this file. (If **Everyone** is selected, then all people, including the Web login account, have access to the file.)

BROWSER CRASHES WHEN RETRIEVING RESULTS FROM APPLICATION SERVER

In general, for machines with less than 32 megabytes of RAM, the browser will perform very poorly when parsing HTML files that are larger than 150 kilobytes. Therefore, users who do not have at least 32 megabytes of RAM installed on their machines should refrain from retrieving large queries.

An example of a large query follows. Suppose you query Customer by Fiscal Year and apply the filter Business Unit: Learning. This query returns approximately 3800 customers, and the HTML that is generated is 1.8 megabytes. When loaded in Netscape 4.03, it occupies 37 megabytes, takes 3 minutes to parse, and consumes the entire processor. The parsing is the bottleneck in the downloading of the file.

Note: When started, Netscape Navigator 4.03 requires 8 megabytes immediately to load whatever it is loading. Microsoft Internet Explorer (IE) requires 6.7 megabytes, and is substantially faster at parsing the text.

REFRESH PROGRAM FAILS

If the Refresh program fails, the AppServer continues to use the old metadata information. Users will still be able to log in and view the old Web Pages. This is because the Refresh program reads the new metadata into temporary structures that are atomically exchanged with the old structures only if all of the refresh structures were read correctly.

To solve this problem, rerun the **Refresh** program.

APPLICATION LOG FULL ERROR

If you receive this error, you can take the following action to delete all log events.

Note: If you do not want to delete all of the log events, you can also increase the amount of space allocated to the application log. You can also select the Override as Needed option in the Log Properties dialog box.

1. Open Start\Programs\Administrative Tools (Common)\Event Viewer. Select the Application Log under the Log menu item.
2. Select Clear All Events from the Log menu.

When prompted whether you want to save the log files, and whether you really want to delete all of the log events, respond in the affirmative.

EPICENTER MANAGER ERROR CONDITIONS

EpiCenter Manager operations can fail to complete when the disk volume on which the EpiMart database resides becomes full. The AppServer, in addition to its log files, maintains a record of database queries in metadata. The table that contains these records is used by the aggregate optimizer to select candidate tables and columns for aggregates. If this table grows to the point at which it fills the disk volume, requests to alter the metadata that you make with EpiCenter Manager can fail with a variety of error messages from your database server.

Refer to “Tracking of Queries for Aggregate Optimization,” on page 371 for information on how to resolve this problem.

005270-072500

GLOSSARY

A

Actual Tables

The Actual tables are metadata created by the Adaptive Schema Generator after it has built or modified tables in EpiMart.

These tables are consulted after a schema generation to determine delta operations to perform. Other tools examine them to ensure that the EpiMart schema matches the current metadata definition.

Ad Hoc Query

An improvised query that is composed and executed as the direct result of a user request rather than being scheduled with other batch operations.

Aggregate

An aggregate is a pre-computed summary of an EpiMart table that is designed to improve query response time.

Aggregate Builder (Aggbuilder)

Aggbuilder is the executable program (**agg.exe**) that builds aggregate tables in EpiMart.

Normally, the execution of this program is scheduled after all data has been extracted from source systems and merged into EpiMart.

Aggregate Instruction

An aggregate instruction is a specification of a single aggregate to be built on a fact table. The aggregate builder executable is used to build the aggregate specified by an aggregate instruction. 165

Aggregate Navigation

The process by which the E.piphany query machinery determines the optimal aggregate table for addressing an end-user query.

Aggregate Optimizer

The Aggregate Optimizer is a tool that helps database administrators construct optimized sets of aggregates based on user-query logs.

Application Server

The Windows NT Service that connects with a Web server and serves up E.piphany Web pages and reports. An Application Server is configured to connect with an EpiCenter.

Approximate Count

A statistical method for estimating the number of members in a list, which takes far less time than counting.

AppServer

See Application Server.

AppServer Monitor

A Web-based interface for monitoring and refreshing the Application Server.

Attribute

Attributes, which are derived from columns in dimension tables, appear as items that end users can select on Web pages. Attributes also serve as filters that users apply to results to refine (drill down) their queries.

Attribute Factory

An EpiCenter Manager tool for creating transaction filter filters (two-occurrence filtering). Filters are stored as attributes, so creating a new filter creates a new attribute, hence the term Attribute Factory.

B

Backfeed Table

A backfeed table is an EpiMart table that holds data about campaigns that have been run. Data can be extracted from a backfeed table for use in the EpiCenter.

Backlog Type

The backlog type indicates that a measure term is to display accumulated data. BEGIN indicates accumulation through the start of a reporting period. END indicates accumulation through the end.

Base Dimension

A base dimension is a physical dimension table in EpiMart. Base dimension tables contain the actual dimension values. Base dimensions are defined globally across an EpiCenter.

Base Dimension Aggregate

A base dimension aggregate is a table in EpiMart that represents an aggregated view of a base dimension table.

Base Table

Base tables are the EpiMart tables (both fact and dimension) that store nonaggregated data at extraction granularity. They are used as input to Aggbuilder to build aggregate tables. Base table names end in the _0 suffix.

Behavior Component

One portion of the specification for a link behavior. A link component indicates whether or not to carry state or automatically create a report.

C

Calendar

A Web page type that displays scheduled campaigns.

Campaign

A campaign is the specification for a set of output files that each contain a treatment code and a list of recipients. When a campaign is run, a query creates and populates the data files.

Campaign Manager

An E.piphany module that captures data related to your current marketing program. Reported items include campaign name, expected revenue, type of mailing, and so forth. When you send targeted mail, you have run a campaign.

Carry State

The behavior component that specifies that a link carries state information forward into the Web page at the destination node.

Clusters

Clusters are natural groupings that occur within your data. The Community Clusters Web page type identifies such clusters. *See also*, Fact Clusters.

Community Clusters

A Web page type that identifies groupings within sets of attributes.

Cumulative Projections

A Web page type that projects cumulative results for a current time period based on previous time periods and current results so far.

Current Table

An EpiMart fact table that contains recently-extracted data. Current tables have suffixes of _A or _B.

Custom Fact Index

A custom fact index defines indexes, in addition to Date, to build for a fact table.

D

Data Store

A data store is a logical location from which data is extracted to, or downloaded from, an EpiCenter. Data stores include relational database connections, files, and directories.

Data Store Role

In an extraction job, a data store role specifies how a selected data store is to be used in the job.

Data Warehouse

A data warehouse transforms the raw data from an organization's source-system databases into a star-schema format that is optimized for end-user ad hoc query and analysis.

Database Schema

A database schema is a structure for organizing source data into tables through which information can be accessed by end users in a concise format, such as reports, charts, lists, and campaigns.

Date Dimension

The Date dimension is a special base dimension table supplied by E.piphany that stores all attributes related to time. All fact tables receive the **date_key** foreign key into this table.

Degenerate Dimension

A degenerate dimension is a textual field, such as a bill of lading number, stored directly in the fact table, as opposed to standard dimensions, for which the fact tables store foreign keys rather than the data itself.

Demographic Base Dimensions

The demographic base dimension tables for individuals and groups are used by the List Manager and Campaign Manager to generate lists of individuals and groups.

Demographic Dimensions

Individual and group dimension roles associated with the **Ind_Group_Joiner** table.

Demographic Filters

Filters that are based on attributes of the **group** or **indiv** dimension roles.

Dependency

A dependency is a relationship between two queues that prevents them from being executed at the same time. If Queue B depends on Queue A and both queues are ready to be executed, then the scheduler executes Queue A before Queue B.

Dimension Column

A dimension column is a single column or attribute of a base dimension table. It contains the actual customer data that appears on reports or in filters.

Dimension Column Access

Dimension column access is the ability for a user or group to view only a subset of the available data in an EpiCenter. Dimension-column-access settings can limit specific users to data for only one region.

Dimension Column Set

A dimension column set is a named set of dimension columns (all within a single base dimension) that defines which columns are used in a base dimension aggregate.

Dimension Role

A dimension role is a logical view of a base dimension table that allows data from that table to appear in reports of differing scope.

Dimension Tables

Dimension tables contain attribute data, such as the names of customers and territories. Attributes are represented as global objects in an EpiCenter.

E

E.piphany Application

An E.piphany application is a set of one or more Web pages and a topic master that provides a map for navigating among those pages to solve a class of business problems.

E.piphany Solution

An E.piphany solution is an instance of an E.piphany application that has been installed and configured for a particular organization. 32

EpiCenter

An EpiCenter is a single logical E.piphany datamart that includes customer data and control metadata. It physically consists of two databases: EpiMeta and its associated EpiMart.

E.piphany Confidential

EpiCenter Manager

E.piphany's versatile program for configuring, administering, re-configuring, and updating your database. You can use its graphical user interface to define your EpiMeta database table schema.

EpiCenter Manager is also the means by which you configure Web pages for end users, set up the jobs needed for data extraction, and perform administrative tasks.

EpiChannel

EpiChannel is the E.piphany program (**extract.exe**) that executes extraction jobs. EpiChannel implements the extraction steps that are stored as metadata in the EpiCenter.

EpiMart

EpiMart is the physical database that contains actual datamart data. The schema of tables in EpiMart is determined by the metadata in EpiMeta. The data contents of EpiMart tables are determined by the extraction steps in EpiMeta.

EpiMeta

EpiMeta is the metadata repository for an EpiCenter. It contains all control information for the EpiCenter and therefore defines the behavior of that EpiCenter. EpiMeta points to EpiMart via the special EpiMart data store.

External Table

An external table is a table in any data store that is made available for use by the E.piphany system. External tables are commonly used as intermediate tables in multi-staged extractions.

Extraction

Extraction is the process of copying data from external source systems into an E.piphany EpiCenter.

Extraction Group

An extraction group is an ordered collection of extraction steps.

Extraction Step

An extraction step is a single atomic extraction operation. It can be an SQL statement, a semantic instance, a truncation step, an extraction group, or a system call.

F

Fact Aggregate

A fact aggregate is a physical table in EpiMart that contains aggregated fact information for a single fact table.

Fact Cluster

A fact cluster is a user-defined index into a list manager fact table based on a selected dimension. Fact clusters can significantly improve list manager performance.

Fact Column

A fact column is a single numeric column in a fact table.

Fact Count

A fact count is a special index on list manager tables that allows Oracle database servers to make use of fact clusters.

Fact Table

A fact table is a physical table in EpiMart that contains numeric data and references to dimension tables that specify attributes about that data. Facts are represented as global objects in an EpiCenter.

Fact-Table Cluster

A copy of a fact table that is sorted on an attribute.

Epiphany Confidential

Fact-Table Counts

A record of how often entries for a group or individual appear in a fact table.

Filter Element

A filter element is a single check box or entry within a list box that appears on a Web page.

Filter Group

A filter group is a logical grouping of filter elements.

Filters, Attributes, Objects, and Measures

These are the parameters that a user specifies in a Web page to create a typical report.

FOAM

See Filters, Attributes, Objects, and Measures. 303

Foreign Key

A reference in one relational-database table to a key value that resides in another table.

G

Glossary Entry

A glossary entry provides end users with online definitions for configured elements that appear on Web pages.

Group Campaigns

A Web page type that creates campaigns directed toward groups.

Group Dimension

a special dimension that stores demographic data about groups for lists and campaigns.

Group List Manager

A Web page type used to create lists of groups such as compaines or households.

H

Help Page

A Web page that displays usage information about E.piphany Web pages or general information about E.piphany solutions.

High/Low Clusters

A Web page type that identifies combinations of attribute values associated with highest and lowest values.

History Table

An EpiMart fact table that contains older data that is expected to change infrequently. Historical tables have suffixes of _X or _Y.

Home Page

A Web page that is displayed first when a user logs in.

I

Ind_Group_Joiner

This is a special fact table for lists and campaigns that connects the individual and group base dimension tables.

Individual Campaigns

A Web page type that creates campaigns directed toward individuals.

Individual Dimension

A special dimensions that stores demographic data about individuals for lists and campaigns.

Individual List Manager

A Web page type that is used to create lists of individuals.

Influences

A Web page type that is used to identify predictive relationships among a set of attributes.

Intermediate Web Page Types

Web page types that allow users to fill in missing state information before jumping to a node that requires that state in order to automatically generate a report.

J

Job

A job is a top-level workflow object that defines a sequence of data extraction steps to be performed by EpiChannel. It also specifies the data stores for that execution.

Job Step

A job step is a single unit of execution in an extraction job.

L

Lifecycles

A Web page type that projects the life cycle for a new product based on previous like products.

Link

A hypertext link that allows a user to jump from one E.piphany Web page to another. Links have associated behaviors that determine if state is to be carried forward and if a report is to be generated automatically.

Link Category

A set of links that appear together on a Web page. The link category determines where a link appears on the Web page at the originating node.

List

A list is a set of demographic row keys that is created by an end-user query and is saved in the Report Gallery.

List Manager

An E.piphany module that allows end users to generate lists related to individual and group demographic data. Other Web pages can utilize lists as filters.

Login Page

A Web page displayed by the AppServer that allows users to confirm their identities before gaining access to an E.piphany application.

M

Measure

A measure is a single business calculation. It can be an arithmetic combination of measure terms using RPN (Reverse Polish Notation).

Epiphany Confidential

Measure Mapping

The mapping of measures to Web page selections.

Measure Set

A measure set consists of one or more measures used together (as a set) that apply to an Influences or Community Clustering Web page. Measure sets are associated with one of three types: Classification, Regression, or Clustering.

Measure Term

A measure term is a single component of a measure. It can be combined with other measure terms to create a composite measure (business calculation).

Modeling

A Web page type that identifies predictive relationships for use in scored lists.

MomentumBuilder

MomentumBuilder is the executable (**momentumbuilder.exe**) that creates the special tables that are used to generate lists and campaigns.

N

Navigation Node

A navigation node is a location within a topic to which a Web page is assigned. A user performs an activity in the page at that node, then follows links to other nodes.

Navigation Type

The link-behavior component that indicates the window in which the Web page at a destination node is to be displayed. This window could be the same as the origin, a pop-up, or another browser window.

News Banner

An area of the Home page that you can customize to display a daily message or an applet.

O

Object Gallery Pane

The Object Gallery is a pane in an EpiCenter Manager dialog that allows you to drag available objects and drop them into the appropriate configuration categories.

P

Primary Dimension

A primary dimension is a dimension that is the target of a data-mining analysis.

Profiling

A Web page type that charts comparative values.

Proxy

An ISAPI application that mediates I/O between the Web server and the E.piphany Application Server.

Pull/Push SQL Statement

A pull/push SQL statement is a type of extraction step that issues SQL against an input data store and then pushes the result set into a table in the output data store.

Q

Query Machinery

The component of the E.piphany Application Server that communicates with EpiMart and issues SQL statements against the DBMS.

Queryable Column

A queryable fact or dimension table column is a column that can be used in a List Manager or Campaign Manager query.

Queue

A queue is a collection of tasks that can be executed by the scheduler. The scheduler executes queues according to their dependencies. Within a queue, tasks are executed sequentially based on their assigned priority level.

R

Report

A report is a saved set of Web-page settings. Users can save and retrieve reports in the Report Gallery. Saved reports do not include results.

Report Gallery

A component of E.piphany's front-end user interface that allows users to save and retrieve reports, lists, and campaigns. 165

Rows and Columns

The Basic Rows and Columns Web page type displays measure data that is classified across a pair of attributes. Advanced Rows and Columns allows nested drill-down reports.

S*Schedule*

A Web page type that allows users to schedule campaigns.

Scheduler

The Scheduler is the E.piphany subsystem that schedules and executes automated tasks, such as extraction jobs and campaigns.

Schema Generator

The Schema Generator is a component of EpiCenter Manager that builds the tables in EpiMart using metadata definitions in EpiMeta.

Scoring

A Web page type that allows users to rank the members of a list according to a predictive model or a measurement.

Scrutiny

Scrutiny is a debugging tool that identifies inconsistencies in the metadata that specifies your EpiCenter configuration.

Seed

A seed is a predetermined recipient that is used to validate campaigns. The IndivSeed and GroupSeed tables can contain entries for seeds.

SeedJoiner Fact Table

The SeedJoiner fact table is a built-in fact table that must be populated to list all seed records.

EpiPhany Confidential

Select a Navigation Path Web Page Type

A Web page type that allows users to choose from among several links, but does not execute a report.

Semantic Instance

A semantic instance is an SQL program that applies business rules to extracted data. It is an instantiation of a semantic type.

Semantic Template

A semantic template is an SQL template for a semantic type.. When the template is instantiated as a semantic instance, the resulting SQL refers to actual columns and tables.

Semantic Type

A semantic type is a logical business process that is implemented in a semantic template.

Solution Map

A solution map is a typical series of steps that a user might take to solve a particular type of business problem. *See also* Topic Master.

Source System

A source system is a repository of business data, such as an OLTP system, from which data is extracted into an EpiCenter datamart.

SQL Statement

A SQL statement is an extraction step that issues custom SQL against an input data store. The results of the SQL statement can either be discarded or used to push data into a table in the output data store.

Staging Table

A staging table is an intermediate table in an EpiCenter that holds source system data in preparation for merging into the datamart.

00625513 072500

Stand-Alone SQL Statement

A stand-alone SQL statement is a SQL statement whose results are discarded.

Star Schemas

A dimensional data warehouse uses a star schema to organize tables. At the center of a standard star schema is a fact table that contains measure data.

Radiating outward from the fact table like the points of a star are multiple dimension tables that contain attributes. In an E.piphany star schema, the stars are interjoined.

State Information

The accumulated set of reporting parameters that have been selected in current and previous Web pages, which can be carried forward to the Web page at a destination node.

System Call

A system call is an extraction step that causes an operating system program to be invoked.

T***Tab***

A screen or area within an EpiCenter Manager dialog box that is accessed by clicking the image of a tab near the top of the dialog box.

Task

A task is an activity that can be scheduled in a queue. Tasks include such things as extraction jobs and campaigns.

Task instance

A task instance is a single scheduled or completed execution of a task.

Topic

A topic is a named set of navigation nodes, the Web pages assigned to each node, the links between each node, and the behaviors associated with each link.

Topic Master

A template for a topic that includes a predefined set of navigation nodes, links, and behaviors but omits specific Web pages.

Transaction

An action that an individual or a member of a group might participate in, such a purchase, a return, a call to a call center, the receipt of a treatment, or a response to a campaign.

Transaction Filter

A transaction filter allows a user to filter dimension demographics by measure data. Transaction filters are associated with a fact table, a transaction type, and a set of attribute filters.

Transaction Filter Filters

A transaction filter filter, which is defined via EpiCenter Manager's Attribute Factory allows two-occurrence filtering.

Transaction Type

Transaction types distinguish rows with different interpretations in the same fact table. For example, an Order fact table might contain both a BOOK and SHIP transaction type.

Transaction Type Set

Multiple transactions can be grouped as a set and applied to a single measure term. Applying a transaction type set allows the use of a single calculation (usually SUM) for all rows for all transaction types in the set.

Trends

A Web page type that identifies trends along a set of attributes and projects expected values based on those trends.

U

UNKNOWN Dimension Row

The **UNKNOWN** dimension row is a special row in every dimension table that is used as a referent by all fact table rows that do not refer to an actual dimension element.

By convention, the string **UNKNOWN** is generally used to refer to this dimension row.

V

View List

A Web page type that allows users to view and download data pertaining to the current membership of a list.

W

Web Page

A component of the E.piphany user interface that is displayed by a Web browser. An E.piphany Web page allows a user to request reports, lists, and campaigns, and to navigate to other Web pages.

Web Page Type

A prototype for a Web page that specifies the overall appearance of the page, the types of reports that a user can request, and the types of reusable data elements that can appear.

00525510.072500

INDEX

A

- A and B tables, 77, 97, 98
 - browsing aggregates for, 207
 - toggling, 76
- About this Application dialog box, 153
- Access rights
 - group/user precedence, 274
 - user/group precedence, 279
- Actual tables, 53
- Actuals metadata, exporting, 493
- Adaptive architecture, 52
- Adaptive Schema Generator, 60, 433, 493
- \$\$ADD_DAYS macro, 390
- \$\$ADD_MACRO macro, 390
- \$\$ADD_MONTHS macro, 390
- Administrative rights, 276
- Administrator group, 377
- Advanced Rows and Columns Web page type, 316
- \$\$AGG macro, 412
- Aggbuilder, 47, 63, 81, 110
 - behavior of, 52
 - log file structure, 110
 - program steps, 87
- Aggregate building
 - extraction phase, 63
 - invoking, 110
- Aggregate Instruction dialog box, 206
- Aggregate instructions, 202
- Aggregate navigation, 47, 362
- Aggregate Optimizer, 89, 174
 - clearing logs, 371
- Aggregate Optimizer Dialog box, 174
- Aggregate star, 84
- Aggregate tables, 82
- Aggregates
 - browsing built, 187
 - browsing fact, 207
 - cumulative projections web page type, for, 88
 - defining dimension columns for, 172
 - dimension, 82, 175
 - dimension, recommendation, 184
 - effect on Web pages, 319
 - fact, 82, 179
 - High/Low Clusters web page type, for, 89
 - influences web page type, for, 89
 - optimal number of, 173
 - optimization, 89, 174
 - profiling web page type, for, 88
 - purpose of, 82
 - vs. custom fact indexes, 198
- Aggregation, 82
- agg_timestamp, Aggbuilder log directory, 110
- agg.exe command, 110
- all_groups_table, 113
- all_individuals_table, 113
- Allow Anonymous authentication, 375
- \$\$ANALYZE_TABLE Oracle macro, 408
- Application Log Full Error, 503
- Application Server, 44
 - aggregate navigation, 362
 - aggregates at query time, 87
 - aggregates rebuilt, 363
 - cannot log in, 496
 - error messages, 496
 - GIFs, 499
 - log files, 361, 369
 - log naming conventions, 369

proxy logging, 381
 refresh time, 365
 refreshing, 363
 Registry keys, 378
 registry keys, 378
 restarting, 363
 running from command line, 368
 security, 371
 setting up new domain, 376
 starting/stopping Windows NT
 service, 359
 verifying operation, 360
 Windows Registry, 362
 Approximate counts, 326
 \$\$APPSEVERHOST macro, 412
 AppServerHost registry key, 378
 AppServerLogVerbosity registry
 key, 378
 \$\$APPSEVERPORT macro, 412
 AppServerPort Registry key, 378
 AppServerQueryTimeout registry
 key, 378
 AppSessionTimeout registry key, 378
 AppStateTimeout registry key, 378
 Arithmetic operators, RPN, 251
 \$\$ASSERT_INDEX_EXISTS
 macro, 390
 Attribute dialog box, 264
 Attribute Factory, 270
 Attributes, 39
 defining, 264
 removing duplicate, 140
 Attributes filter types, 266
 Authentication
 basic, 376
 modules, 374
 Pass through, 374
 Windows NT domain, 374

B

Backfeed, 63
 Backfeed Dimension semantic
 type, 453
 Backfeed tables, 60
 extraction from, 436, 442
 mirroring in, 98
 Background image, 336
 Backlog types, 250
 Base Dimension dialog box, 165, 170,
 190
 list filter option, 167
 selecting physical types, 427
 Base dimension staging tables
 populating, 435
 queries with joins, 434
 SQL statements, 431
 Base dimension tables, 37, 85, 161
 defining, 165
 BASE HREF tag, 499
 Base tables, 85
 Basic authentication, 376
 Basic Rows and Columns Web page
 type, 316
 \$\$BATCH_PARALLEL_DEGREE
 macro, 391
 \$\$BEGIN_ASSERT_INDEX
 macro, 391
 Behavior, of links, 303
 \$\$BIGDATE macro, 391
 Binning, 434
 Bookings, 455
 \$\$BOOL_TO_YN macro, 391
 Breakpoints, setting, 117
 SQL, 223

C

- Calendar Quarters, 295
- Calendar Web page type, 320
- Calendars
 - fiscal quarters, 49
 - 4-4-5 quarters, 49
- Campaign dimension table, 162
- Campaign Manager
 - built-in dimension roles, 188
 - built-in dimension tables, 162
 - Cell base dimension table, 162
 - Communication fact table, 163
 - extraction for, 435, 436, 442
 - security, 169
 - seed records, 162
 - seeding, 163
 - SeedJoiner fact table, 163
 - tables for, 94
- Campaign Output, 128
- Campaign output format, 283
- Campaign queue, 164, 238
 - scheduler, 164
- Campaign-list pulls, 229
- \$\$CASE_BEGIN macro, 391
- \$\$CASE_ELSE macro, 391
- \$\$CASE_ELSEIF macro, 392
- \$\$CASE_END macro, 392
- \$\$CAT macro, 392
- \$\$CBIN_END macro, 392
- \$\$CBIN_VAL macro, 392
- Cell dimension table, 162
- Channels, 286
- \$\$CHARTSLOGFILE macro, 412
- \$\$CHARTSOUTPUTDIR macro, 412
- ChartsOutputDir registry key, 378
- \$\$CHAR_1 macro, 392
- Check Boxes filter type, 266
- Child queue, 123
- chn_timestamp, EpiChannel log
 - directory, 110
- Choose EpiCenter dialog box, 147, 151
- Choose Groups dialog box, 281
- Classification measure sets, 258
 - defining, 260
- Classification tree, Influences, 259
- Cleansing tools, used by external
 - tables, 76
- Clustered index, fact table, 196
- Clustering, 168
- Clustering measure sets, 258
- Clusters
 - defining for List Manager, 95, 200
 - fact-table, 328
 - guidelines, 200
- Colors, setting for SQL, 153
- Column-access restrictions, 169
- \$\$COLUMN_CURRENT_VALUE
 - macro, 385
- \$\$COLUMN_FILTER macro, 385
- \$\$COLUMN_LAST_VALUE
 - macro, 386
- \$\$COLUMN_RANGE_FILTER
 - macro, 386
- Commands
 - agg.exe, 110
 - epiappservice.exe, 365–367
 - EpiCenter, 153
 - EpiMgr, 153
 - Export All, 493
 - extract.exe, 218
 - Generate Schema, 53, 54
 - Help, 153
 - jview.exe, 368
 - momentumbuilder.exe, 112
 - Object, 153
 - Presentation, 153
 - Refresh View, 155
 - refresh.exe, 363
 - regedit.exe, 495
 - Server, 153
- Communication fact table, 163, 191
- Community Clusters Web page
 - type, 168, 316
- com.epiphany.security.EpiNTLogon
 - security module, 372

Condense Dimension Roles dialog
 box, 142
 config_master table, 417
 Configuration dialog box, 160, 248,
 295, 326, 417, 500
 COUNT DISTINCT operators, 251
 unique ID column, 168
 Count measure roles, 259, 261
 Count Unjoined semantic type, 62, 79,
 465
 \$\$COUNTER macro, 393
 \$\$COUNT_ROWS_FROM
 macro, 393
 \$\$COUNT_ROWS_SELECT
 macro, 393
 Counts
 defining for List Manager, 96
 fact-table, 328
 guidelines, 200
 List Manager, 200
 Create Targeted Fact Aggregate dialog
 box, 180
 \$\$CREATE_INDEX_IF
 NOT_EXISTS macro, 393
 Cumulative Projections Web page
 type, 320
 aggregates for, 88
 \$\$CURR macro, 388
 Currencies
 multi-currency option, 421
 system usage, 421
 Currency measurement, 248
 current_datamart value, 363
 \$\$CURREXP macro, 388
 \$\$CURRHIST macro, 388
 \$\$CURRVIEW macro, 389
 Custom fact indexes, 82, 92, 93, 94, 198

D

Data merging extraction phase, 60, 77
 Data Store dialog box, 211, 213
 Data store roles, 80
 Data stores, 64
 assigning, 228
 creating, 211
 EpiMart, 66, 213
 generic ODBC, 213
 input and output, 210
 JobFileLog, 66, 213
 LoggingDB, 66, 213
 Microsoft SQL Server, 213
 Oracle, 213, 407
 Data warehouses, 31, 32
 Database administration tools, 58
 \$\$DATABASE macro, 412
 Database server, registering, 147
 DatabaseName registry key, 379
 DatabasePassword registry key, 379
 DatabasePort registry key, 379
 DatabaseServer registry key, 379
 DatabaseType registry key, 379
 DatabaseUsername registry key, 379
 Datamart mirroring, 97
 Datamart tables
 current, 187
 toggling, 136, 299
 Date Dimension Dialog box, 151
 Date dimension table, 38, 49
 fields, 423
 populating, 151, 293, 295
 Date range, EpiMart, 295
 \$\$DATE_CURRENT_VALUE
 macro, 386
 \$\$DATE_FILTER macro, 386
 date_key, 293, 439
 \$\$DATE_LAST_VALUE macro, 387
 date_modified column, 433
 \$\$DATE_RANGE_FILTER
 macro, 387

- Dates, granularity in dimension roles, 192
- date_type, 418
- Date, displayed on reports, 500
- Date.fy_name, 279, 282
- \$\$DBNOW macro, 393
- \$\$DBVENDOR macro, 412
- \$\$DDL_BEGIN macro, 394
- \$\$DDL_BEGIN_NO_DECLARE macro, 394
- \$\$DDL_END macro, 394
- \$\$DDL_EXEC macro, 395
- Debug level, job, 217
- \$\$DEBUG macro, 389
- Debugging. *See* Scrutiny
- \$\$DEBUG_LEVEL macro, 412
- \$\$DECLARE_BEGIN macro, 395
- \$\$DECLARE_BODY macro, 395
- Default job log file, directory for, 149
- Degenerate Dimension dialog box, 190
- Degenerate dimensions, 53, 190
- Dependencies of queues, 121
- Description registry key, 379
- Dialog boxes
 - About this Application, 153
 - Aggregate Instruction, 206
 - Aggregate Optimizer, 174
 - Attribute, 264
 - Attribute Factory, 270
 - Base Dimension, 165, 170, 190, 427
 - Choose EpiCenter, 147, 151
 - Choose Groups, 281
 - Condense Dimension Roles, 142
 - Configuration, 160, 248, 295, 326, 417, 500
 - Create Targeted Fact Aggregate, 180
 - Data Store, 211, 213
 - Date Dimension, 151
 - Degenerate Dimension, 190
 - Dimension Column, 166
 - Dimension Column Access, 278, 282
 - Dimension Role, 189, 190
 - Execute Job, 217
 - Exporting Metadata, 297
 - External Column, 240
 - External Tables, 427
 - Fact Column, 194
 - Fact Table, 193, 194, 198
 - Fill From Query, 270
 - Find Report, 290
 - Generate Schema, 293
 - Glossary, 256, 262, 267
 - Importing Metadata, 297
 - Initialize EpiCenter, 148, 149
 - Job, 214, 217
 - Matching Dimension
 - Aggregates, 178
 - Matching Fact Aggregate, 183
 - Measure, 247, 252, 255
 - Measure Layout, 255
 - Measure Sets, 260
 - New Group, 203
 - Output Format Entry, 284
 - Output Processor, 286
 - Populates fact table option, 438
 - Preferences, 153
 - Preview Dimension Aggregate, 177
 - purpose of description field, 155
 - Queue, 231
 - Register EpiCenter, 147
 - Report, 158
 - Semantic Instance, 226
 - Server Properties, 147
 - SQL Macro, 241
 - SQL Statement, 221, 431
 - SQL Statement, Tables References option, 432
 - Transaction Type Sets, 209
 - Truncate Steps, 227
 - User, 280
- Dimension aggregates, 82, 87, 170
 - recommendation for, 184
 - trade-offs, 173
- Dimension Column Access dialog box, 278, 282
- Dimension Column dialog box, 166
- Dimension columns
 - displaying values of, 282
 - restricting access to groups, 277

restricting access to users, 282
 UNKNOWN, 167
 Dimension Role dialog box, 189, 190
 Dimension roles, 37
 condensing, 143
 custom fact index, 199
 foreign keys, 187
 group and indiv, 323
 Dimension semantic types, 79, 447–453
 Dimension tables
 aggregation of, 39
 defining dimension roles, 189
 number required, 161
 queryable columns in, 94
 Dimensions, attributes of, 39
 \$\$DIMINDEX_TABLESPACE Oracle
 macro, 408
 DimRoleName_sskey, 440
 \$\$DIMTABLESPACE Oracle
 macro, 408
 \$\$DIRNAME macro, 413
 Domains, 376
 \$\$DOUBLESTRING macro, 395
 \$\$DROP_INDEX macro, 396
 \$\$DROP_TABLE_IF_EXISTS
 macro, 396
 \$\$DSN macro, 413
 Duplication feature, 155
 Dynamic Check Boxes filter type, 266
 Dynamic filter types, 268
 Dynamic Listbox filter type, 266

E

\$\$ELSE macro, 396
 E-mail
 configuring for job status, 243
 determining profile name, 243
 EpiCenter Manager default
 address, 244, 417
 job status notification, 216

Microsoft Outlook Exchange, 244
 password, 417
 verifying notification, 244
 End of Extraction, 76, 110, 500
 \$\$END_ASSERT_INDEX macro, 396
 \$\$END_IF macro, 396
 end_year, 418
 Entire Dimension filter type, 266
 \$\$EOS macro, 396
 epiappservice command, 365–367
 \$\$EPIBIN macro, 413
 EPIBIN system call macro, 81
 EpiCenter command, 153
 EpiCenter macros, 388
 EpiCenter Manager, 47, 57, 133
 cloning tables, 155
 configuration, 158
 default job, 214
 defining base dimension tables, 165
 duplication feature, 155
 error conditions, 504
 Generate Schema, 293
 invoking commands, 152
 measure terms, 209
 registering server, 147
 starting the program, 146
 window, 154
 EpiCenter Manager Report, 158
 EpiCenter toolbar icons, 154
 EpiCenters, 57
 appropriate number of, 134
 group-only, 197
 initializing, 148
 producing new, 298
 registering existing, 151
 unregistering, 150
 EpiChannel, 47, 58, 59, 67, 81, 101–120
 debugging levels, 116
 error messages, 116
 logging, 119
 output, 114, 118
 See also extract command
 selecting debug level, 222
 trial-run model, 118

- verbose debugging option, 116
- verbosity levels, 217
- EpiChannel log file, 450, 451, 454
 - sskey rows, 448, 452
- \$\$EPIINT macro, 396
- \$\$EPIKEY macro, 397
- EpiMart, 35, 57, 64
 - building tables in, 53
 - customer data, 47
 - data range, 295
 - data store, 66, 213
 - dimension column values, 282
 - initial load of, 452
 - schema changes in, 52
 - table-naming conventions, 53
 - tables, 47
 - using Scrutiny to test, 299
- EpiMart tables, 47
 - base, 85
 - building, 53
 - mirroring in, 96
 - naming conventions, 53
 - purging, 245
 - toggling A and B, 76
- EpiMeta, 35, 57, 66
 - cloning, 493
 - tables, 46
- EpiMgr command, 153
- EpiNTLogon, 376
 - Windows NT authentication module, 372
- E.piphany
 - Application Server. *See* Application Server, 357
 - authentication modules, 374
 - database schema, 31
 - macros *See* Macros., 384
 - system overview, 44
- Epiphany.dll proxy, 358, 380, 381
- Error conditions, EpiCenter Manager, 504
- Error messages
 - Application Log Full, 503
 - Application Server, 496
 - EpiChannel, 116
 - Object Not Found, 500
 - Refresh program fails, 503
 - Registry Editor warning, 495
 - User Cannot Log In, 496
- \$\$EXC macro, 413
- \$\$EXC_ARGS macro, 413
- \$\$EXC_CMD macro, 413
- \$\$EXEC_SP macro, 397
- Execute Job dialog box, 217
- Export All command, 493
- Export machinery, 492
- Export tables, 494
- Exporting Metadata dialog box, 297
- export.mdb, 138, 298
- External Column dialog box, 240
- External tables, 60, 76, 239
 - cleansing tools, 76
 - input to staging queries, 443
 - last_extract_date, 76, 239
 - physical types for, 427
- Extraction
 - aggregate building phases, 63
 - campaign manager, 111
 - data merging phase, 60, 77
 - list manager, 111
 - load phase, 59, 77
 - memorized values, 109
 - multi-stage, 79
 - new rows, 106
 - setting up for EpiCenter, 210
 - stages, 59
- Extraction Date Unknown, 500
- Extraction Groups, 69
 - End of Extraction, 76, 110, 500
- Extraction macros, 385
- Extraction statements, 76
- Extraction steps, 68
 - semantic instance, 77
 - shared by job steps, 71
 - SQL statement, 71
 - system call, 79

INDEX: F

extract.exe command, 67, 102, 218
 command-line arguments, 103
 directory location, 81
 See also EpiChannel
e.4 End of Extraction, 76, 500

F

Fact aggregate instructions, 136
Fact aggregates, 82, 87
 browsing, 207
Fact Column dialog box, 194
Fact columns
 defining, 193
 Queryable option, 194
Fact semantic types, 79, 453–465
Fact staging tables, loading, 463
Fact Table dialog box, 193, 194, 198
Fact tables, 39
 clustered index, 196
 clusters in, 95
 counts in, 96
 defining, 191, 193
 queryable columns in, 94
 reloading populated, 462
 transactional format of, 50
\$\$FACTINDEX_TABLESPACE Oracle
 macro, 408
\$\$FACTMONEY macro, 397
\$\$FACTQTY macro, 397
Facts
 in EpiMart, 39
 rows with zero facts, 453
 state-like, 62
 transactional, 62
Fact-table clusters and counts, 328
\$\$FACTTABLESPACE Oracle
 macro, 408
\$\$FILENAME macro, 414
Fill From Query dialog box, 270
Filter elements, 268

Filter groups, 268
Filters, attributes, options, and measures
 (FOAM), 303, 306
Filters, demographic, 324
Find Report dialog box, 290
First Dimension Value semantic
 type, 61, 451
First/Last Fact semantic type, 62, 463
Flat files, source system, 65
\$\$FLOAT macro, 397
Foreign keys, 38
 and primary keys, 48
 degenerate dimension, 197
 fact tables, 195
 to date dimension, 49

G

General-purpose macros, 390, 407
Generate Schema command, 53, 160,
 293
 schema altered, 54
Generate Schema dialog box, 293
GIFs, generated by Application
 Server, 499
Global objects, defining for
 extraction, 220
Glossary dialog box, 256, 262, 267
Glossary entries, 262
Group attributes usage contrasted with
 Individual, 329
GROUP BY statements, 432
Group Campaigns Web page type, 320
Group List Manager Web page
 type, 323
Groups, setting up, 274

H

Help command, 153
 High/Low Clusters Web page type, 322
 aggregates for, 89
 Historical partitions, 61
 History tables, 96, 97
 Home page, 312

I

\$\$IDENTITY macro, 397
 \$\$IF macro, 397
 IIS security, 375
 IIS Web server. *See* Web server
 \$\$IJ_FROM macros, 397
 ikey, 455
 Importing Metadata dialog box, 297
 Incremental jobs, 99, 218, 486
 \$\$INCREMENTAL_PARALLEL_DEGREE macro, 398
 Indexed for fast search option, 167
 Indexes
 fact table, 92, 93
 RDBMS, 49
 Indexing, 82
 custom fact, 198
 fact table clustered index, 196
 leading term, 196
 Ind_Group_Joiner fact table, 326
 Individual Campaigns Web page type, 320
 Individual List Manager Web page type, 323
 Influences
 classification tree, 259
 regression tree, 259
 Influences Web page type, 316
 aggregates for, 89
 Initial jobs, 99, 218, 486

Initial Load Dimension semantic type, 61, 78, 449, 452
 invalidation of saved lists by, 452
 Initial Load Fact A/B semantic type, 62, 462
 Initial Load Fact semantic type, 62, 78, 462
 Initialize EpiCenter dialog box, 148, 149
 \$\$INITIAL_LOAD macro, 389
 \$\$INSTANCE_NAME macro, 414
 \$\$INSTANCEROOTDIR macro, 414
 InstanceRootDir registry key, 379
 \$\$INSTR macro, 398
 Internet Server Application Programming Interface (ISAPI), 358
 Inventory data, transactional system, 52
 ISAPI *See* Internet Server Application Programming Interface
 ISQL, 500
 iss, 448

J

Job dialog box, 214, 217
 Job steps, 69
 ordering, 214
 Job types, 99, 218, 486–489
 incremental, 99, 486
 initial, 99, 486
 merge, 100, 487
 non-partitioned, 100, 488
 rebuild, 100, 487
 JobFileLog data store, 66, 213
 JobLogFile, 213
 \$\$JOB_NAME macro, 414
 Jobs, 114
 data store roles, 80
 debug level for, 217

default, 215
 disabling, 216
 e-mail notice of status, 243
 monitoring, 119–120
 role of, 67
 \$\$JOIN_LEFT_OUTER macro, 398
 \$\$JOIN_RIGHT_OUTER macro, 398
 \$\$JOIN_WHERE macro, 399
 jview command, 368

L

last_extract_date, 76, 239, 500
 Latest Dimension Value semantic type, 61, 449
 Leading terms, fact table indexing, 196
 \$\$LENGTH macro, 399
 Life Cycles Web page type, 323
 Lifecycles Web page type, 322
 Link, 303
 List Manager
 extraction, 242
 installing, 148
 removing, 148
 security, 169
 tables for, 94
 List Manager tables, 94
 List Membership filter type, 266
 Listbox filter types, 267
 Load extraction phase, 59, 77
 Log database, 110, 119
 Log device, 119
 Logging Data Store, 213
 LoggingDB data store, 66, 213
 logging_mssql.sql script, 213
 Logging, proxy, 381
 Login page, 314
 Logs
 location of, 119
 role of, 118

\$\$LOJ_FROM macro, 399
 \$\$LONGSTRING macro, 399

M

Macros

EpiCenter, 388
 extraction, 385
 general, 390–407
 Oracle, 408
 semantic instance, 69
 SQL, 73, 433
 SQL Server, 409
 SQL, syntax of, 384
 system-call, 412
 system-call, syntax of, 411
 Mail Profile name, 244, 417
 \$\$SMARTDBNAME macro, 389
 Matching Dimension Aggregates dialog box, 178
 Matching Fact Aggregate dialog box, 183
 MaxDBConnections registry key, 379
 \$\$MAX_SYS_DATE macro, 399
 Measure dialog box, 247, 252, 255
 Measure Layout dialog box, 255
 Measure layouts
 defining for Web page, 254
 removing redundant, 136
 Measure mapping
 importing, 298
 instructions, 254
 verifying, 257
 Measure roles
 Count, 259
 SumSquared, 259
 TargetSum, 259
 Measure sets, 258
 defining Classification, 260
 defining for Community Clustering Web pages, 261

defining for list membership, 260
 defining for regression, 261
 types of, 258
 Measure Sets dialog box, 260
 Measure terms, 209, 248
 Measure units, 248, 420
 defining, 420
 symbols for, 420
 Measures, 246
 defining, 247
 mapping elements to, 257
 Merge jobs, 100, 218, 487
 Metadata, 35
 clearing query-log tables, 371
 converting to Release 4.0, 135
 current state of, 294
 exporting individual objects, 296
 exporting/importing of, 296
 primary keys in tables, 491
 steps for importing, 138
 \$\$METADBDNAME macro, 389, 399
 \$\$METATABLESPACE Oracle
 macro, 408
 Microsoft Access
 database, 296, 298
 database tables, 491
 Export database, 494
 Microsoft Internet Explorer, 503
 Microsoft Outlook Exchange, 244
 Microsoft SQL Server
 data store, 213
 Physical type values, 427
 TCP/IP sockets, 498
 Mirroring, datamart, 97
 Modeling Web page type, 316
 \$\$MODULO macro, 400
 MomentumBuilder, 111, 112
 command line, 112
 default system call, 113
 verifying extraction, 113
 momentumbuilder.exe. See
 MomentumBuilder
 MomentumOutputDir registry key, 379
 \$\$MONEYSTRING macro, 400

Monitors, 217
 MSSQL Server SQL Service, 363
 Multi-currency option, 421

N

Navigation nodes, 334
 granting access to, 277
 \$\$NBIN_END macro, 400
 \$\$NBIN_VAL macro, 400
 Netscape Navigator, 503
 New Group dialog box, 203
 \$\$NEXT macro, 389
 \$\$NEXTEXP macro, 389
 \$\$NEXTHIST macro, 389
 \$\$NO_FROM_LIST macro, 400
 Non-Partitioned jobs, 100, 219, 488
 \$\$NOT_DEBUG macro, 389
 \$\$NOT_\$\$INITIAL_LOAD
 macro, 389
 NTLM authentication, 376
 NT. See Windows NT
 \$\$NUMBER(9,5) macro, 400
 \$\$NVL macro, 401

O

Object command, 153
 Object Gallery, 156
 Object Not Found, 500
 ODBC
 data store, 213
 layer, 65
 source system, 65
 fiscal_year start_m, 418
 Oracle
 data store, 213
 macros, 408

INDEX: P

Physical type values, 429
tablespaces, 407, 409
\$\$ORACLE macro, 401
Output Data Store, 212
Output format, 283
Output Format Entry dialog box, 284
Output Processor dialog box, 286
Output processors, 286

P

P and Q tables, 98
 See also A and B tables.
Parent queue, 123
Pass-through authentication, 374
\$\$PASSWORD macro, 414
\$\$PATH macro, 414
Percent measurement, 248
Permissions
 default for new users, 279
 folder and report, 292
Physical types, 427
 selecting for fact column, 194
 values for Oracle, 429
 values for SQL Server, 427
Pipelined semantic type, 62, 460
Populates a dimension option, 432
Populates fact table option, 438
Preferences dialog box, 153
Presentation command, 153
Preview Dimension Aggregate dialog
 box, 177
Primary dimension, 168, 188
Primary group membership, 281
Primary keys, 48, 491
 correspondence to sskeys, 433
process_key, 439, 455, 457, 461
Profiling Web page type, 333
 aggregates for, 88
\$\$PROGRAM_NAME macro, 414

ProxyLogFile registry key, 379
proxy.log file, 379

Q

Quarter Projections Web page
 type, 320
Quarters 4-4-5, 151, 295
Query machinery, 47, 63
Query performance, improving, 82
Query run time, setting, 281
Queryable columns, 94
Queryable option, fact column, 194
QueryCacheSize registry key, 379
Query-log tables, clearing from
 metadata, 371
Queue dialog box, 231
Queues, 121
 blocked, 126
 child, 123
 circular dependencies in, 124
 default, 122
 dependencies, 121
 killing of dependent, 122
 logs, 129
 parent, 123
 running, 125
 sleeping, 125
 tasks in, 121
 transitivity of dependencies, 123
 waiting, 125

R

Radio Buttons filter type, 267
\$\$RAISE_EXCEPTION macro, 401
RDBMS
 physical types for platform, 427
 server, 64, 210

Rebuild jobs, 100, 219, 487
 Refresh View command, 155
 refresh.exe command, 363
 fails, 503
 regedit.exe command, 495
 Register EpiCenter dialog box, 147
 Registry Editor warning, 495
 Registry keys, 104, 378
 Application Server, 378
 SecurityClass, 372, 375
 \$\$REGISTRY_EPIPATH macro, 414
 Regression measure sets, 258
 defining, 261
 Regression tree, Influences, 259
 Reload Date Fact A/B semantic
 type, 62, 463
 Reload Date Fact semantic type, 62,
 463
 \$\$REMOVE_MACRO macro, 401
 \$\$RENAME_OBJECT macro, 401
 Report dialog box, 158
 Report Gallery, 274, 287, 314
 Reports
 copying/moving, 290
 date displayed on, 500
 e-mailing, 280
 folders for saved, 289
 Result Page Error, 500
 Reverse Polish Notation, 252
 arithmetic operators, 251
 translation of, 251
 Rewrite Dimension semantic type, 61,
 453
 Rows and Columns Web page
 types, 316

S

SAM. *See* Security Access Monitor
 Saved list, invalidation, 452
 Saved reports

and object name changes, 156
 default permissions, 292
 upgrading, 142
 Schedule Web page type, 320
 Scheduler, 121, 229
 configuration parameters, 132
 log files, 131
 logs, 129
 operation of, 125
 running as console application, 132
 setting up, 230
 Schema
 changes in EpiMart, 52
 definitions, 54
 generating, 293
 operations, 54
 Scoring, measures for, 252
 Scrutiny debugging tool, 299
 ScrutinyDisabled registry key, 379
 Security
 access rights, 273
 authentication, 273
 column-access restrictions, 169
 Epiphany system, 273
 Windows NT groups, 274
 Security Access Monitor, 375, 377
 Security Manager, 372
 SecurityClass Registry key, 372, 375,
 380
 SecurityManager log, 370
 Seed dimensions, 162
 extraction for, 435, 442
 Seed records, 163
 SeedJoiner fact table, 163, 192
 Select a Navigation Path Web page
 type, 315
 SELECT DISTINCT query, 435
 SELECT statements, 437
 \$\$SELECT INTO BEGIN
 macro, 402
 \$\$SELECT INTO BEGIN_OPT Oracle
 macro, 408
 \$\$SELECT INTO BEGIN_TS
 macro, 408

- \$\$SELECT_INTRO_BODY macro, 402
- Semantic Instance dialog box, 226
- Semantic instances, 53, 58, 60, 69, 75
 - defined, 78
 - defining, 226
- Semantic templates, 60, 78, 433
- Semantic transformation, 77
- Semantic types, 60, 78
 - Backfeed Dimension, 453
 - Count Unjoined, 62, 79, 465
 - defined, 78
 - dimension, 79
 - fact, 79
 - First Dimension Value, 61, 451
 - First/Last Fact, 62, 463
 - Initial Load Dimension, 61, 78, 449, 452
 - Initial Load Fact, 62, 78, 462
 - Initial Load Fact A/B, 62, 462
 - Latest Dimension Value, 61, 449
 - Pipelined, 62, 460
 - Reload Date Fact, 62, 463
 - Reload Date Fact A/B, 62, 463
 - Rewrite Dimension, 61, 453
 - Slowly Changing Dimensions, 61, 447, 451
 - Transactional, 62, 454
 - Transactional/State-Like, 62, 455
 - Transactional/State-Like/Force Close, 62, 456
- Semantics, 57, 58
- Server command, 153
- \$\$SERVER macro, 414
- Server Properties dialog box, 147
- Shipments, 455
- Slowly Changing Dimensions semantic
 - type, 61, 169, 319, 447, 449, 451, 452
 - effect on Web pages, 319
- \$\$SMALLDATE macro, 402
- \$\$SMALLINT macro, 402
- Solution maps, 36, 42, 353
- Source system identifier keys. *See* sskeys
- Source system keys. *See* sskeys
- SQL
 - dialect problems, 223
 - sample for job step, 221
 - sample for populating a table, 224
 - syntax colors. setting, 153
- SQL Macro dialog box, 241
- SQL macros, 73, 409, 433
 - creating new, 241
 - NVL, 432
 - syntax of, 384
- SQL operators, 249
- SQL Server macros, 409
- SQL Statement dialog box, 221
 - sample SQL, 224
 - Tables References option, 432
 - Template feature, 431, 432, 434
- SQL statement extraction steps, 71
- SQL statements, 69
 - error in, 72
 - fact staging, 437
 - populating staging tables, 223, 431
 - push/pull, 71
 - setting breakpoints, 117, 223
 - stand-alone, 71
- \$\$SQLNET macro, 414
- \$\$SQLSERVER macro, 402
- \$\$SSKEY macro, 402
- sskeys, 73, 74, 433, 440, 448, 450, 451, 454, 455, 456, 462
 - duplicate, 434
 - entering a pipeline stage, 461
 - ikey, 448, 455
 - new rows, 448
 - resolving for different databases, 212
 - ss key, 439
 - UNKNOWN row, 74
- Staging tables, 57, 59, 73, 75
- Star schemas, 57, 161, 434
- start_day_445, 418
- start_year, 418
- StateDir registry key, 380
- State-like facts, 62, 455
- Static filter types, 269
- StorageManager log, 370

\$\$\$SUBSTRING macro, 403
 Sum measure roles, 261
 SumSquared measure roles, 259, 261
 \$\$\$SUPERNVL macro, 403
 SVGA monitors, 217
 Symbol character, measure unit, 420
 Synchronize option, adding users, 275
 Synchronized groups, 376
 System call objects, defining, 226
 System calls, 79
 exit code, 81
 System-call macros, 412
 syntax of, 411
 SystemLogDir registry key, 380
 SystemLogDirWebpath registry
 key, 380

T

\$\$TABLE_EXISTS_CONDITION
 macro, 403
 Tablespaces, Oracle, 407
 alternate names for, 409
 Tables, rebuilding all, 293
 \$\$TABLE_WITH_PREFIX
 macro, 403
 TargetSum measure roles, 259
 Task instances, 127
 completed, 127
 expired, 127
 overdue, 127
 Tasks, 121
 expired, 127
 instances, *See* task instances.
 logs, 130
 overdue, 127
 scheduling of, 126
 TCP/IP sockets, SQL Server, 498
 TempDirGarbageLifetime registry
 key, 380
 TemplateDir registry key, 380

\$\$TEMP_TABLESPACE Oracle
 macro, 408
 Text Box filter types, 267
 Time dimension (Date_0), 48
 Time zone, displayed on reports, 500
 \$\$TIMESTAMP_FILTER SQL Server
 macro, 409
 \$\$TIMESTAMP_FILTER_RANGE
 SQL Server macro, 410
 \$\$TIMESTAMP_LAST_VALUE SQL
 Server macro, 410
 Timestamps, 106, 107, 109, 119
 ignore option, 107, 217
 Time, treatment of, 48
 \$\$TINYINT macro, 403
 \$\$TO_CHAR macro, 404
 \$\$TO_CHAR_UNIVERSAL
 macro, 404
 \$\$TO_DATE macro, 404
 \$\$TO_DATEFMT macro, 404
 \$\$TO_EPIDATE macro, 404
 \$\$TO_HHMMSS macro, 405
 \$\$TO_INT macro, 405
 \$\$TO_NUMBER macro, 405
 Topic masters, 303
 Topics, 303
 granting access to, 277
 Toplevel template, 375
 \$\$TO_TIME macro, 405
 \$\$TO_YYYYMMDD macro, 405
 Transaction filter filters
 creating, 270
 duplicates, 142
 Transaction filters, 270, 325
 Transaction keys, reserved, 419
 Transaction type dimension, 38
 Transaction Type Sets dialog box, 209
 Transaction type sets, defining, 209
 Transaction types, 162, 439
 EpiCenter, 419
 multiple, 192
 problems with renaming, 493
 Transactional facts, 62

Transactional semantic type, 62, 454
 Transactional/State-Like semantic type, 62, 455
 Transactional/State-Like/Force Close semantic type, 62, 456
 Transactions
 EpiMart fact tables, 50
 source system data as, 52
 tracking changes in inventory, 51
 \$\$TRANSLATE_BEGIN macro, 405
 \$\$TRANSLATE_ELSE macro, 405
 \$\$TRANSLATE_END macro, 405
 \$\$TRANSLATE_VAL macro, 405
 \$\$TRANSTYPE macro, 406
 TRANSTYPE SQL macro, 493
 transtype_key, 439, 457, 460
 transtype_0 table, 420
 Trends Web page type, 333
 Truncate Steps dialog box, 227
 Truncation objects, 227

U

Units measurement, 248
 UNKNOWN
 dimension row, 73, 74
 rows, 299
 string, 74, 432
 \$\$UNKNOWN_DATE macro, 406
 User dialog box, 280
 \$\$USER macro, 415
 Users, setting up, 279

V

\$\$VAR macro, 406
 \$\$VAR_ASSIGN_BEGIN macro, 406
 \$\$VAR_ASSIGN_END macro, 406

\$\$VAR_ASSIGN_INT macro, 406
 \$\$VARCHAR macros, 406
 Verbose option. EpiChannel, 116
 \$\$VERSION macro, 415
 Version number, EpiCenter Manager, 419
 VGA monitors, 217

W

Web browser, 287
 login dialog box, 376
 Web pages, 44, 303
 assigning access to, 282
 glossary entries, 262
 HTML version, 41
 performance issues, 320
 Web server, 44, 358, 380, 499, 501
 authentication, 372
 authentication method, 372
 finding alisas, 499
 instance_name alias, 499
 log file location, 380
 starting, 366
 stopping, 366
 URL address, 358
 Web server error message, Object Not Found, 500
 week_start_day, 419
 Windows NT
 authentication, 280
 authentication module, 371, 372
 domain authentication, 374
 group access rights, 274
 group member synchronization, 372
 importing users into groups, 276
 Performance Monitor, 119–120
 Working directory, 110, 118
 location of, 119

X

X and Y tables, 97
browsing aggregates for, 207
see also A and B tables

Y

Years, beginning/end of EpiMart, 295
Yen sign, 420
\$YYYYMMDD_CURRENT_VALUE
macro, 387
\$YYYYMMDD_FILTER macro, 387
\$YYYYMMDD_FILTER_RANGE
macro, 388
\$YYYYMMDD_LAST_VALUE
macro, 388

09625518.072500